Contexts and user modeling through disentangled representations learning

Perrine Cribier-Delande

SCIENCES SORBONNE UNIVERSITÉ

THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ
Spécialité **Informatique**
École Doctorale Informatique, Télécommunications et Électronique (Paris)

Contexts and user modeling through disentangled representations learning

Apprentissage de représentations démêlées pour la modélisation de contextes et d'utilisateur

Présentée par
Perrine Cribier-Delande

Dirigée par
Ludovic Denoyer

Co-encadrée par
Vincent Guigue

Pour obtenir le grade de
DOCTEURE de SORBONNE UNIVERSITÉ

Présentée et soutenue publiquement le

Devant le jury composé de :

Latifa Oukhellou *Directrice de Recherche, IFSTTAR – GRETTIA*	Rapportrice
Liva Ralaivola *Professeur, Criteo Research Lab*	Rapporteur
Patrick Gallinari *Professeur, Sorbonne-Université – LIP6*	Examinateur
Thierry Artières *Professeur, École Centrale Marseille – LIS*	Examinateur
Vincent Guigue *Maitre de Conférence, Sorbonne-Université – LIP6*	Co-encadrant de thèse
Ludovic Denoyer *Professeur, Facebook Research*	Directeur de thèse

ABSTRACT

The recent, sometimes very publicised, successes have drawn a lot of attention to Deep Learning (DL). Many questions are asked about the limitations of these techniques. The great strength of DL is its ability to learn representations of complex objects.

Renault, as a car manufacturer, has a vested interest in discovering how their cars are used. Learning representations of drivers is one of their long-term goals. Renault's strength partly lies in their knowledge of cars and the data they use and produce. This data is almost entirely contained in the Controller Area Network (CAN). However, the CAN data only contains the inner workings of a car and not its surroundings. As many factors exterior to the driver and the car (such as weather, other road users, road condition...) can affect driving, we must find a way to disentangle them.

Seeing the user (or driver) as just another context allowed us to use context modelling approaches. By transferring disentanglement approaches used in computer vision, we were able to develop models that learn disentangled representations of contexts. We tested these models with a few public datasets of time series with clearly labelled contexts. Using only forecasting as supervision during training, our models are able to generate data only from the learned representations of contexts. They even learn to represent new contexts, only seen after training.

We then transferred the developed models on CAN data and were able to confirm that information about driving contexts (including driver's identity) is indeed contained in the CAN.

RÉSUMÉ

Les récents succès, parfois très médiatisés, de l'apprentissage profond ont attiré beaucoup d'attention sur le domaine. Sa force réside dans sa capacité à apprendre des représentations d'objets complexes.

Pour Renault, obtenir une représentation de conducteurs est un objectif à long terme, identifié depuis longtemps. Cela lui permettrait de mieux comprendre comment ses produits sont utilisés. Renault possède une grande connaissance de la voiture et des données qu'elle utilise et produit. Ces données sont presque entièrement contenues dans le CAN. Cependant, le CAN ne contient que le fonctionnement interne de la voiture (rien sur son environnement). De nombreux autres facteurs (tels que la météo, les autres usagers, l'état de la route...) peuvent affecter la conduite, il nous faut donc les démêler.

Nous avons considéré l'utilisateur (ici le conducteur) comme un contexte comme les autres. En transférant des méthodes de désenchevêtrement utilisées en image, nous avons pu créer des modèles qui apprennent des représentations démêlées des contextes. Supervisés uniquement avec de la prédiction pendant l'entrainement, nos modèles sont capables de générer des données à partir des représentations de contextes apprises. Ils peuvent même représenter de nouveaux contextes, qui ne sont vus qu'après l'entrainement (durant l'inférence).

Le transfert de ces modèles sur les données CAN a permis de confirmer que les informations sur les contextes de conduite (y compris l'identité des conducteurs) sont bien contenues dans le CAN.

REMERCIEMENTS

Je tiens tout d'abord à remercier mes encadrants qui m'ont guidée au cours de ces quatre années parfois compliquées. Merci à Ludovic de m'avoir donné l'opportunité de réaliser cette thèse et de m'avoir accompagnée pendant les première années de cette recherche et soutenue dans tous les moments difficiles. Merci à Vincent d'avoir repris un peu (beaucoup) au pied levé la direction et de m'avoir permis de terminer cette thèse alors que les choses ne se présentaient pas très bien. Enfin, merci à Raphaël, pour son soutien sans failles tout au long de cette thèse ainsi que son aide pour appréhender les problématiques industrielles.

Je souhaiterais aussi remercier Latifa Oukhellou et Liva Ralaivola pour le temps et l'énergie investis dans la rédaction de rapports très enrichissants sur mon manuscrit ainsi que leurs remarques et questions pertinentes qui m'ont permis de prendre du recul sur mon travail. Mes remerciements vont aussi aux autres membres de mon jury Patrick Gallinari et Thierry Artières pour leur lecture attentive de mon manuscrit et les retours constructifs qu'ils m'ont faits.

Chez Renault, je souhaiterais remercier toute l'UET "connectivité" avec une mention particulière pour Anthony Vouillon dont la bienveillance et la bonté m'ont permis de passer trois années au sein d'une équipe très agréable. Les discussions passionnées avec mes collègues (en particulier Jean-Guillaume Meyrignac et Raphaël Puget) m'ont permis d'apprendre de moi et du monde et ont rendu un quotidien parfois difficile beaucoup plus supportable.

Pour le LIP6, je souhaiterais remercier toute l'équipe MLIA (ancien.ne.s et nouve.lles.aux, certains déjà cités) qui ont fait de cette thèse beaucoup plus qu'un travail. Il y en a trop pour toustes les citer, mais les doctorant.e.s qui sont passé.e.s dans notre équipe m'ont toustes apporté quelque chose d'important. Beaucoup d'entre elleux sont devenu.e.s des ami.e.s.

Je souhaiterais aussi remercier Christophe Bouder pour son aide inestimable et ses réactions quasi immédiates à mes problèmes techniques parfois triviaux (parfois non). Nadine Taniou est aussi une composante indispensable de notre travail et je la remercie pour sa bienveillance et son aide précieuse pour comprendre et naviguer au travers des méandres de la logique de l'administration.

Ma reconnaissance va aussi à Marie-Jeanne Lesot pour son soutien qui a parfois été le dernier fil me reliant à cette thèse. Durant cette dernière année faite surtout de rédaction, de pandémie et de confinement, elle a été mon radeau dans la tempête.

Je ne peux pas oublier l'occupant spécial de la salle 26-00/522 dont la présence inébranlable est une partie incontournable de notre cohésion d'équipe.

Une mention particulière pour les doctorants (et surtout doctorantes) dont le soutien et les conseils au cours de cette thèse m'ont permis de tenir, d'avancer puis de finir : Aurélia, Jean-Yves, Gaby, Marie et Michael.

Doctorante de l'équipe, mais tellement plus que ça, je ne saurais exprimer avec des mots ma reconnaissance à celle qui est maintenant quasiment de la famille, Clara *you changed me for good* !

Last but not least, je souhaiterai remercier ma famille pour m'avoir supporté et soutenu dans cette période difficile durant laquelle mon attitude a parfois été assez peu agréable. Au-delà de cette thèse, leur présence et leur soutien m'a permis d'arriver jusqu'ici. Sans l'héritage social de mes parents, cela est sans doute injuste, mais je n'aurais probablement pas fait cette thèse. Leur apport tout au long de ma vie sur le plan scientifique, mais aussi philosophique, politique et moral m'ont donné une précieuse culture et une vision de la vie qui m'y ancre. Enfin, je souhaite aussi remercier mes frère, sœur et belles-sœur sans lesquelles ma vie serait infiniment plus triste. Leur passion pour les jeux et leur empressement à jouer dès que le besoin s'en fait sentir m'ont permis de rester relativement saine dans un monde qui ne l'est pas.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMS

AE	Autoencoder
AI	Artificial Intelligence
CNN	Convolutional Neural Network
ConvNet	Convolutional Neural Network
DL	Deep Learning
GAN	Generative Adversarial Network
NLP	Natural Language Processing
SVM	Support Vector Machine
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean-Squared Error
NN	Neural Network
RNN	Recurrent Neural Network
DTW	Dynamic Time Warping
AR	Autoregressive
RL	Reinforcement Learning
IL	Imitation Learning
GMM	Gaussian Mixture Model
LSTM	Long Short-Term Memory
CAN	Controller Area Network
OBD	On-Board Diagnostics
GPS	Global Positioning System
RPM	Revolutions Per Minute
DBI	Driver Behaviour Inventory
DBQ	Driver Behaviour Questionnaire
MDSI	Multidimensional Driving Style Inventory
DSQ	Driving Style Questionnaire
DAS	Driver Assistance System
ADAS	Advance Driver Assisance System
IMU	Inertial Measurement Unit
EV	Electrical Vehicule

HEV	Hybrid Electrical Vehicule
PHEV	Plug-in Hybrid Electrical Vehicule
PerClos	Percentage of Eye Closure
ETS	Eye-Tracking Signal
Radar	RAdio Detection And Ranging
LiDAR	Light Detection And Ranging

CHAPITRE 1

INTRODUCTION

Contents

1.1 Context and goals of this work

In the last decade, the rise of data-driven approaches has led to an exponential growth of the domain of Machine Learning (ML) domain. This is especially true for the subdomain of Deep Learning (DL). The many advantages of DL have come to light and advances are still ongoing in many domains : state-of-the-art performances were achieved in Natural Language Processing (NLP), signal processing, computer vision, recommender systems... The uncanny ability of DL models to solve problems by fitting arbitrarily complex functions has led to many real-world impressive applications such as automatic image captioning or colourisation of images. The efficiency of DL over traditional algorithms comes from its ability to learn representations, even of complex objects. The Convolutional Neural Network (CNN) thus revolutionised the domain. The core of the CNN that allowed them to become so popular is their ability to learn intermediate representations. The succession of layers building on each other, each one learning more complex features than the previous one, is their might. Indeed, the first layers detect lines and arcs and angles, that the next layers turn into more complex shapes which are finally combined by the last layers to obtain a representation of real-world objects. The fact that CNNs can be fine-tuned to tackle almost any task in computer vision makes them the universal tool to use when working with images. In another domain, models such as word2vec or fasttext have allowed great progress in NLP. Their ability to represent words as vectors allow other models to use them for tasks as diverse as machine translation, text summarisation or question answering.

The emergence of DL has also driven the renewal of Reinforcement Learning (RL). Indeed, barely a decade ago, the Reinforce algorithm [**reinforce**] was the state of the art, achieving good results but with limited perspectives. Today, agents trained with complex RL models (many using CNNs) have defeated humans in a list of games that keeps getting longer. In 2015, agents were achieving superhuman performance in arcade games [**mnih2015humanlevel**]. Later came the game of Go, maybe the most publicised

achievement of RL, with AlphaGo beating human grand masters [**alphago**]. Modifications to AlphaGo turned it into AlphaStar which then achieved high human-level performances in the very complex video game Starcraft [**Vinyals2019GrandmasterLI**]. All of these achievements would not have been possible without the immense power of representation learning brought forth by DL.

Industrial Motivation The recent, sometimes very publicised, successes have drawn a lot of attention to DL. Many questions are raised about the limitations of these techniques. Are there many possible applications of these techniques we just haven't thought of yet? Almost every industry and service has been researching the possible applications for their area of expertise.

As this work was realised in partnership with Renault, its motivation originated from an industrial issue. The idea was to conduct preliminary studies to determine how these techniques could be used to bring value to the company. The goal was to determine if the expert domain knowledge of Renault and its well-honed skills could be used in combination with data-driven approaches to create new opportunities. This thesis aims at combining the advantages of Renault's knowledge of the data used and produced by car with the DL techniques and their power of representation.

The Wonders of the Controller Area Network (CAN) Bus Designing, thinking and manufacturing cars for over a century, Renault has accumulated a vast knowledge which gives the company an incontestable advantage over its concurrent. For instance, one of the areas in which Renault has a decisive advantage on newcomers is their in-depth knowledge of the Controller Area Network (CAN) bus. The CAN is a standard in the automotive industry that allows the different components of cars to communicate without using a central unit. It appeared in the '80s and was widely used by the early '90s. It represents a shift in paradigm in cars. Before the CAN, controls (such as the wheel or the pedals) were almost directly linked to the part of the car they control. Turning the wheel had a direct effect on the axles, pressing pedals would bring more (or less) gas to the engine and/or increase/decrease the flow of air. The CAN changed this by stepping in between commands and their effects. Instead of directly turning the wheels, the wheel control unit transmits to the wheels control unit a message indicating by how much the wheel was turned. The wheels control unit then translates this message into a set of instructions to obtain the requested heading. In a way, the CAN brought cars out of the analogue era into the numerical one. As time passed, the different control units of cars were optimised. Instead of simply translating the received command, some calculations are made as to how the required effect can be achieved in the optimal way. For example, a command from the acceleration pedal to increase the speed is translated into instructions to no less than 5 effectors in the engine.

In some senses, autonomous cars are the natural next step for the industry. Starting from cars that were controlled directly by mechanical actions we stepped up to cars that are partially computer controlled, then to Advance Driver Assisance System (ADAS) and finally to cars that drive themselves.

From Simulations to Simulators and Data Generation Historically, Renault has always made intensive use of simulation. It has been employed to design cars, enhance their parts and many other purposes. At first, it was mainly used to improve small specific parts of the car and decide which solutions seemed more promising. It is now widely used during the design process of the car, getting its specifications and evaluating its resistance. For example, the safety requirement of cars are now mostly ascertained through simulation and those results are only verified with very few real 'crash tests'. The analysis of cars, their structure, their workings and their design has slowly shifted from physical, empirical analysis to mostly computer analysis and design. Renault's technocentre was built in this objective and is full of people using mostly computers to enhance and design the cars of the future. These many years of work to simulate the inner working of cars has led Renault to develop complex physical models to describe the behaviour of cars with great precision. Indeed, even though cars all obey the same laws of physics, small differences in their design can drastically change their behaviour. Each car model must have its own physical models, which resemble the others but have their own peculiarities. As an example, the friction coefficient with the air of a car is highly dependent on its design and will affect (among many other aspects) the energy needed by the motor to achieve a certain speed.

Because the laws of physics are well defined, these physical models are often computed with high precision through intensive calculations. These models, that can predict the behaviour of the cars in any simulated environment and their response to any stimulation, are then verified through the testing of real cars. Many parameters have to be tested, from the motor response to intensive use in extreme temperatures to the car suspension on damaged or bad roads. Regulations also demand that cars be tested by driving them on long distance in order to measure their resistance and solidity. Some of this testing is done on tracks that were designed for this purpose. They can mimic roads in bad conditions or mountain roads, as well as allowing for speed test in which a car is pushed up to $250\,\mathrm{km/h}$ to test its resistance. The results and measurements of these tests are then fed into the models to verify their accuracy. Oftentimes, some very minor adjustments are needed. The models are adapted to better fit the data, instead of just being the result of exact calculations.

The motivation behind data-driven approaches : taking real use into account From a machine learner's perspective, this is akin to having a complex model with a lot of fixed (or almost fixed) parameters and hyperparameters. These models are extremely useful in many use cases and are currently the base of the autonomous driving research. The use of these models allows the creation of rule-based driving where there is an optimal choice of trajectory to move from one point to another. This is not, in itself, the self-driving engine because it can only compute a short trajectory and the sequence of commands needed to follow it. It does not allow for making decision based on evolving surroundings, it is just a physical view of the world. This limitation of the models also implies that drivers are not taken into consideration at all. Because these models are solely focused on cars and their physics, they do not allow for user modelling or profiling.

On the opposite side, data-driven techniques are very adapted to user modelling. However, they may not be as good as the previous models to perform exact simulations of car behaviour. Indeed, the huge amount of domain expert knowledge needed to develop these models would be extremely difficult to blend into ML models. One domain in which they could prove very useful for Renault is driver modelling.

Renault has a vested interest in understanding how clients use and drive their cars, which physical models and simulations cannot predict. For example, Renault is interested in questions such as how fast drivers wear out their cars, how much gas they consume or how safe or accident-prone they are. All this information can only be derived from the real use of cars. This requires data about actual usage of cars 'in the real world'.

From this data, a relatively simple solution would be to learn a model for each of these questions : one model to predict the wear and tear of a car, another one for the gas consumption, another one that evaluates the safety of a driving style... However, the strength of DL lies in representation learning. Deep models usually extract features from their input before using them to perform their task such as classification, regression, etc. To best leverage this power, we decided on another approach.

1.2 Methods

Our proposed approach : Learning Representation of Contexts We noted that most questions about driving (and cars use) are highly dependent not only on the driver, but on many exterior factors such as location or weather. This is why we decided to learn representations of driving contexts to solve our task. One of the important factors of driving remains the driver. We wondered whether the user (in this case the driver) could just be considered like any other context.

Our stated goal became to obtain good enough representations of driving contexts (weather, location, drivers...) to be able to predict the trajectory of a car in these contexts.

Though the other contexts are interesting for Renault, their main interest lies in the representation of driving styles. This is why, we decided to combine the other contexts into a single one we called task. We believe that a good enough representation of the driver should be transferable to other driving tasks (as long as we have their representation).

An illustration of the task at hand (simplified) is presented in Figure 1.1. In this example, we have trajectories on a roundabout for three drivers and trajectories on a crossroad for two of these drivers. We wish to predict the trajectory of the third driver on the crossroad. The solution we propose is to learn representations of the contexts. We need the representation of the two driving tasks and the three drivers. From them, we aim to predict the 'missing' trajectory.

The Larger Question of Context Modelling This approach of learning contexts to make prediction is transferable to other research problems. For example, one of the drawbacks of DL (and its main cost) is that it requires huge amounts of data. If we can infer entire

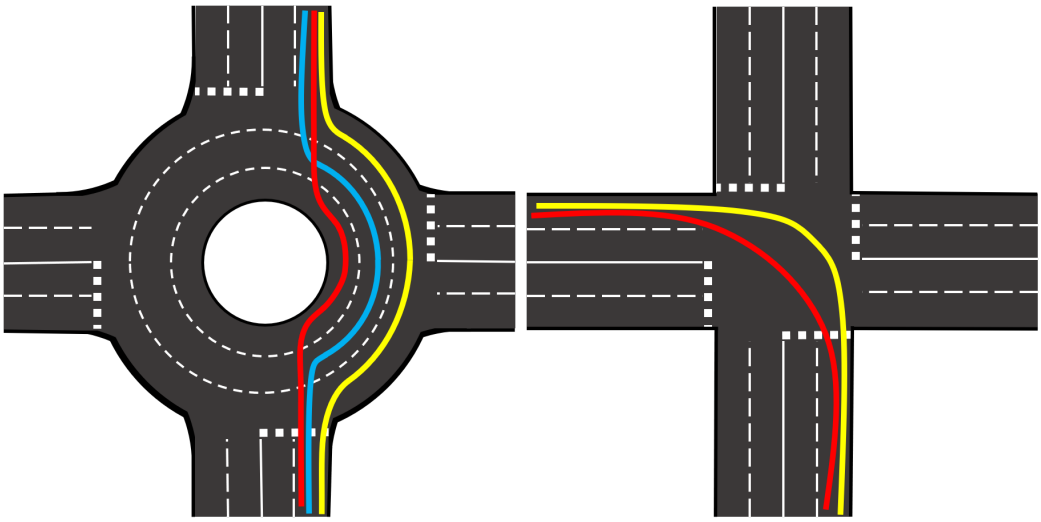

FIGURE 1.1 – We have at our disposal examples of trajectories on a specific driving task (a roundabout), made by three different drivers. Two of these drivers have also driven on another driving task (the crossroad). The goal is to predict the trajectory of the third driver on the second driving task.

signals from only representations of contexts, we should be able to generate missing data from these representations. Therefore, this approach could drastically reduce the cost of collecting data, by diminishing the number of measures to be performed. Data would only have to be collected for a fraction of combinations of contexts and we could infer the signal for the combinations of contexts that have never been measured. Another way of seeing this is to think of conditional generative models as the future of forecasting. Indeed, generation from sufficiently rich contexts and forecasting can be seen as two similar tasks, the information contained in the contexts being enough to predict the first few time steps.

This approach can also be linked to recommender systems and especially the problem of cold start. In collaborative filtering, representations of user profiles and products are learned. With good representations, a simple scalar product between the two gives good recommendations. One of the associated challenge lies in the addition of new products or users. Most of the time, the model needs to be retrained almost from scratch to accommodate new users or products. This is what is referred to as the cold start problem. Our work bridges the two domains by learning good representations of contexts, considering the user as just another context and allowing these representations to be created after training.

On a longer scale, we believe that our work could be used in data augmentation for time series. One of the reasons behind the success of CNNs in computer vision is that their hunger for data could partly be satiated through the use of data augmentation. By slightly modifying the input images without significantly altering their content, new examples can be created, artificially increasing the size of the datasets. This technique does not

have an equivalent in time series because, most of the time, altering the content of a series (without completely changing it) is challenging.

Constraints on Context Representations Many applications where our approach could be used require to learn 'good' representations. By good, we not only mean that representations should contain the necessary information, but also that the latent space in which they exist should be well structured. For example, distance should have a meaning. Two contexts that are close should have close representations. We would also like to be able to sample the space of contexts when sensible. Interpolation between two representations of contexts should give slowly changing contexts going from one to the other. For a context related to weather for example, we should see a smooth transition between rain and sun, passing by drizzle and cloudy. A well-structured space of driving style that is sampleable could be used in many applications.

To obtain such representations of contexts, we decided to use disentanglement techniques. From complex objects (such as multivariate time series), disentanglement aims to obtain representations that are split into parts and explain this separation. Each part corresponds to a real-world concept and does not contain information pertaining to the other ones. For us, it means obtaining representations of the different contexts so that we can change one without changing the other. To draw a parallel with our use case in Figure 1.1, we want to be able to use the blue driver's representation from the roundabout and combine it with the four-way intersections' representation to predict the trajectory.

The Choice of Time Series The CAN data is essentially a multivariate time series. As this domain has known many successes, we decided to focus on time series modelling. Moreover, supervision on representations learning is not easy to obtain. By using auxiliary tasks related to time series, we can avoid the thorny issue of evaluating representation directly. Forecasting, being one of the most research tasks of time series modelling, was our first choice. We also investigate the problem of conditional generation of time series. Indeed, the best way to ascertain that all relevant information is indeed contained in a representation is to attempt to recreate the original example from this representation.

1.3 Organisation

Our main contributions in this work are two-fold :

- A model (or more precisely three incremental versions of a model) that learns representations of contexts from labelled time series. This work has led to the publication of two conference papers :

  esann (Best student paper award)

  itsc

- The adaptation of the previous model to fit CAN data and analysis of the existing work on concepts such as driving style and driving behaviour. As this work was done with an industrial partner, no papers were published (or submitted).

chapitre 2 provides an overview of the state of the art of the techniques that we used in our work.

We investigated the different methods that have been used in time series modelling (both using deep learning or more traditional techniques). We also researched generative models and their modification to make them learn disentangled representations.

chapitre 3 presents the tasks and models that we designed to learn representations of contexts using supervised learning.

We identified two settings where our approach of learning disentangled contexts could be useful. The first setting can be linked to data imputation in the case where data was only collected for some combinations of possible contexts. The goal is to learn representations of contexts to generate the time series corresponding to the other combinations. The second setting, more closely related to the cold start problem, involved new contexts that are discovered after training, during the inference phase. Their representations must be learned 'on the fly' from a unique new time series that matches a new context.

Supervised disentanglement has mostly been used in computer vision tasks, therefore, we had to adapt some techniques so that they could be used with time series. Specifically, our models use representation mixing and adversarial techniques to forecast or generate time series using a representation of contexts. Similar to a classical auto-encoder model, they are comprised of two parts : an encoder and a decoder. The encoder learns to represent disentangled contexts from time series by extracting the salient features of the input. The decoder takes these representations of contexts and attempts to reconstruct the original time series.

Our first model relies on embedding matrices. However, the latent spaces learned by these matrices are notoriously unstructured. They are also discrete by nature, and therefore cannot tackle our second setting of 'on the fly' encoding. By changing the nature of the encoder to a deep neural network, our second model became capable of learning more structured latent spaces. This also allows it to be used in our second setting. Finally, our third model adds an attention mechanism to increase the stability of the model and enhance its resistance to noisy data.

We first tested the proposed models on publicly available datasets with clean data and well-labelled contexts. We choose these datasets because, since they are the aggregation of many daily human behaviours, they already have a strong context component. They also have very well-defined contexts relating to spatiotemporal context. Our models achieved good performances in both forecasting and generation, even compared to models that specialised in these tasks.

chapitre 4 presents the work done on driving style and CAN data.

Our work aimed at exploring the possibilities for data-driven applications for the representation of driving contexts (with a focus on driving style). Much research had to be done on defining fuzzy concepts such as driving style or driving behaviour as well as how to measure them. These concepts do not have an agreed upon definition so we had to decide what each concept would mean to us. We decided to define a driving task as the location and weather associated to a portion of the road. We also made the hypothesis that driving style was unique to one driver and that a mapping exists between the two.

To learn disentangled representations of contexts, we needed to collect data where they are clearly labelled. We therefore designed an experimentation campaign to collect data in a sufficiently controlled environment so that the contexts were clearly defined. The collected data turned out to be overly noisy and needed extensive processing. We performed several preliminary experiments that showed that information pertaining to driving style and task were indeed contained in the CAN. Encouraged by these results, we adapted our models so they could be used with CAN data. Though the results were good, our models were not as effective as with the other datasets. We made a few hypotheses as to why. The first one was that our learned contexts (task and style) are not sufficiently rich. We also based our work on a hypothesis of a mapping between driving style and drivers. This was a first approximation because there are probably different people that have similar driving style. For example, if a parent has taught one of their children to drive, it is likely that they will drive in a similar style. We found that it appears that the driving style of one person can change. Even within a day, depending for example on emotions (and probably other factors) differences can be found. We also think that the CAN data we used might be too noisy (and especially quantified) to provide rich enough contexts. However, despite these issues, we showed that CAN data is a promising avenue for research and insight into drivers' behaviour.

Finally, chapitre 5 will conclude this manuscript and present the research directions we started to explore. The two main extensions we worked on concern the domain of RL and weekly supervised learning.

CHAPITRE 2

RELATED WORK

Introduction

Our goal is to learn good representations of contexts in order to predict behaviours. We decided to use an approach based on time series because our industrial motivation needs them. Time series have been studied with many possible angles such as forecasting, imputation, classification, anomaly detection... and as they are a long-standing research question, methods to model them can be traced back to the 19^{th} century.

We decided to see our task as time series forecasting and generation. This provides a natural supervision that is still much needed when using most deep learned approaches. This supervision is not easily found because we focus on representation learning, a domain where supervision is not as intuitive as in forecasting or classification. Forecasting has been studied for a long time and many algorithms have been developed. Many of these algorithms do not aim at learning representations but only focus on prediction. They only learn representations as a by-product of their training. Nonetheless, our models must achieve better performance in forecasting to prove their usefulness. The generation task is less prominent than the forecasting tasks but it has received a lot of attention in the last few years, though mainly in the computer vision domain. The supervision is less obvious and the task is generally not as well defined. The task of generation ensures that the learned representations are correct, in the sense that they contain all the needed information to reconstruct examples. Moreover, conditional generation is especially interesting to us if the condition can be made on contexts. This would mean that at least part of the learned representations can be interpreted. As representing context separately is our goal, we will focus on techniques that allow it. Disentanglement is a relatively new domain that has mostly been studied in computer vision. Our goal is to transfer these existing techniques to the domain of time series.

In this chapter, **??** will present an overview of methods to model time series, from the historical perspective using simple representations and experts' knowledge to the more recent advances using Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) and almost no domain knowledge. **??** will then present classical algorithms of representation learning, starting with the relatively simple models of dimensionality reduction and finishing with generative models as the evolution of the representation models. These models aim only at representation learning and do not perform it as a means to an end. Finally, **??** we will present previous work on disentanglement and conditional generative model, starting with the task of source separation and ending with recent advances using deep (usually generative) model to learn disentangled latent spaces.

2.1 Time Series Modelling

The statistical analysis of time series has been a challenge for a very long time [**Yule1927**] and forecasting them is an even older problem as the review by **history** showed. Indeed, the very wide range of possible applications in modelling time series lead many researchers to focus on this domain. Applications range from economics to smart cities, from

weather forecasting to physical processes and many more. In this work, we used modelling mostly to obtain a supervision and to ensure that our learned representations are good. We mostly performed forecasting and generation with a few tasks of classification.

Formally, for a time series $\mathbf{x}$, if we denote x_t the series at index t and $\mathbf{x}_0^{t-1}$ the $t-1$ first time step of the series, time series forecasting can be defined as finding the model f such that :

$$x_t = f(\mathbf{x}_0^{t-1}) + \epsilon_t \tag{2.1}$$

where ϵ_t is the one-step ahead forecast error of x, also called the shock of the series at time t. It is also the error made by the forecasting model and should ideally be 0. These models can be used whether or not the time series are of the same length or not.

2.1.1 Historical approaches

Because these problems have been studied for a very long time, some methods to solve them have been used for many years. We will focus on the main approaches of forecasting and classification : the Autoregressive (AR) and Moving average (MA) models for forecasting, the tree-based models for forecasting and classification and a variety of models specific to classification. Many of these models use intermediate representation to perform their task. Because these representations are often key to successful classification and forecasting, we will then present the methods used to construct alternate representations.

2.1.1.1 Autoregressive and Moving Average Models

The concept of stationarity is central in time series forecasting and is especially important in the case of AR and MA models. Intuitively, it means that the way a time series is generated does not change over time and therefore that if we are able to model the beginning of the series, this model should hold for the rest of it. Formally, a series $\mathbf{x}$ is (weakly) stationary if its mean and variance are invariant under translation, meaning that $\forall t \quad \mathbb{E}(x_t) = \mu$, (where μ is a constant) and $\mathcal{C}\text{ov}(x_t, x_{t-u})$ is only a function of u.

AR models assume that current values are linearly dependant of past values limited to a period. Using **??**, this means that f is a linear combination of a few past values. The number of past values to use is called the period p of the AR model. The parameters of this model are the p weights of the linear combination (plus a bias).

MA models assume that the current value is linearly dependant on the prediction error of our model for the past values. Using **??**, the focus is on the shock ϵ. This time, the prediction is made with a linear combination of the past shock. The period is noted q, which is the number of time steps to look in the past for this shock. The parameters of this model are the q weights of the linear combination (plus a bias).

The Autoregressive Moving Average (ARMA) model is a combination of both. The current value is a linear combination of the p previous values and the q previous shocks. This model has $p + q$ weights (and a bias).

As such, AR, MA and ARMA models are adapted to stationary time series. They also work best when the time series are not correlated and independent of each other. However, time series are not necessarily stationary.

To overcome this shortcoming, Autoregressive Integrated Moving Average (ARIMA) models were created by **box1968some**. This model combined the ARMA model and adds an integrating process (integrating in this case is used as the opposite of differentiating). Indeed, one way to make a series stationary is to differentiate it (subtracting the $t - 1$ time step from the t) and losing the first time step in the process.

The ARIMA model takes three hyper-parameters : p, the lag order (how far in the past to look for data) d the degree of differentiation (the number of times the raw observations are differentiated) and q the moving average (how far to look in the past for shock) :

$$f(x_{t-1}) = c + \sum_{i=1}^{p} \phi_i w_{t-i} - \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{2.2}$$

with

$$w_t = x_t - x_{t-d} \tag{2.3}$$

where the ϕ and $\boldsymbol{\theta}$ are the learned parameters of the model (the AR part and the MA parts respectively).

Many other variations exist around ARIMA models, like SARIMA (that takes into account seasonality), ARFIMA (that performs more sophisticated differentiation) and also factorisation of all these models (adaptation of these models for multivariate time series). We will not get into more details here. The advantage of models such as the ones presented here is that, once built, forecasting is quite easy. The disadvantage of these models is that they assume linear relations between time steps while some series do not hold this property.

2.1.1.2 The Prophet tool

The forecasting of time series using models relying on autocorrelation function and parametric models (such as AR models) represents only one approach of time series analysis. When the series cannot be approximate by a linear one, for example when studying climate change, other approaches have to be considered. One such class of models worth noting are based on Generalized Additive Model (GAM). The idea is to decompose the model in several components, each corresponding to different features of the time series to model. The well-known Prophet tool introduced by **prophet** is based on such decomposition. Time series are decomposed into three components : one account for periodicity (season, month, year, etc), another accounts for holidays (which are not necessarily yearly) and the last component account for non-periodic changes in the time

series. The Prophet tool helps analysts forecast time series, mostly by a fine-grained use of seasonality that is refinable by the experts.

2.1.1.3 Tree-Based Models

Another class of models used in time series is based on trees. Tree based models have the advantage that they can be used for both forecasting and classification.

The most famous example of tree-based model is probably the random forest [**random_forest**]. This ensemble model combines multiple trees to create a 'forest'.

Trees [**decision_tree**] are supervised models that partition data very efficiently. Starting at the first node, at each step, a single aspect of the data is examined and the process goes forth with another node that depends on the result of this examination. The result (and therefore the node transition) can be stochastic or deterministic. They were initially made by hand with nodes representing very clear decision or examination. However, using data and training, the nodes and the transitions associated with them can be learned. Trees can be used in time series to perform either classification or forecasting.

The main advantage of trees is that they are explainable. Each decision can be understood by following the nodes and examination that led to it. The main disadvantage of the trees is that they are often inaccurate as they grossly overfit the data and fail to capture its complexity.

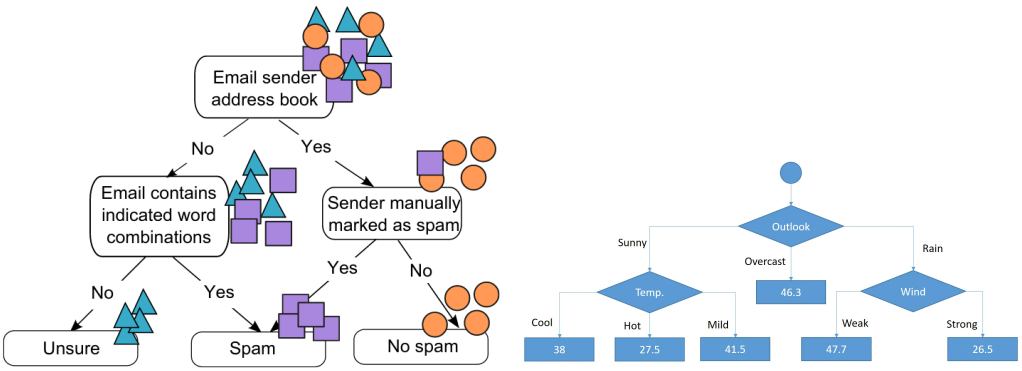

FIGURE 2.1 – Two examples of trees : one for forecasting, one for classification. On the left a decision tree to perform classification of emails. Credits to **decision_tree_image**. On the right a regression tree that predicts the average number of golf players from the weather. Credits to **regression_tree**.

To remedy this problem, **random_forest** presented random forests. Random forests combine multiple trees to mitigate their inaccuracy. This, in turn, renders random forest much less explainable. The idea behind random forest is to learn an ensemble of trees that are uncorrelated. The result of the whole model is based on a 'vote' of these trees. For classification the random forest outputs the class with the majority of votes. For regression, the random forest output an average of all the trees' results.

The method to learn decorellated trees is called bagging, for Bootstrap AGGregatING [**bagging**]. Instead of training random trees on the entire dataset, many subsets of the dataset are sampled (uniformly and with replacement) and a tree is trained for each 'subdataset'. This reduces the correlation between trees and reduces overfitting in each tree. This process reduces the variance of the model without increasing its bias.

If one (or a few features) are strong predictors for the target, the trees will learn to rely on those features and become correlated. This is why, in addition to the dataset sampling of bagging, random forests also use random subspace method (also called feature bagging) presented by **feature_bag**. In this process, features are also sampled (uniformly and with replacement) for each subdataset. This means that each tree is trained on a subset of features of a subset of examples. This further reduces correlation between trees.

These two bagging methods allow reducing the variance of the model and therefore overfitting.

Another tree based algorithm is the Gradient Boosted Decision Trees (GBDT) [**friedman2001**]. Boosting is the method where sequentially connected weak learners are combined to create a strong learner. In this case, the weak learners are trees. The difference with random forest is that instead of the trees being learned in parallel, they are learned in series. Boosting replaces bagging. In boosting, each tree attempts to minimise the error made by the previous trees. This error is measured by a loss function that allows computing residuals. These residuals are used as the loss to train the next tree. This forces each tree to focus only on the examples that are not being handled well (e.g. misclassified examples). Because the trees are learned sequentially instead of in parallel, training is slower than for random forest. The danger of overfitting is also more important than in random forest. Adding too many trees will increase the risk of overfitting, whereas random forests' accuracy don't suffer from more trees. GBDT are, however, more accurate than random forest.

An improvement of GBDT is XGBoost (eXtreme Gradient Boosting) [**Chen_2016**]. According to its creator, Tianqi Chen : 'The name xgboost, though, actually refers to the engineering goal to push the limit of computation resources for boosted tree algorithms. Which is the reason why many people use xgboost.' This algorithm is not fundamentally different from GBDT but it adds hardware optimisation and allows for more parallelisation through distributed computing. It also adds some desirable features that improve stability, training time and accuracy. Amongst them are smart tree pruning and regularisation to decrease the risk of overfitting. It has become very popular in the last few years by winning many machine learning competitions such as Kaggle competitions. At the time of writing, it is the state of the art in many industrial challenges.

2.1.1.4 Other Algorithms Used for Time Series Classification

There are many other classes of model that can perform forecasting or generation. Too many for us to discuss here. In forecasting, the state of the art of non-deep learning methods lies in AR models and XBGoost. We will briefly mention other state-of-the-art methods for time series classification (that do not use deep learning) inspired by **tsc**.

One method that is often used as a benchmark for time series classification is the Dynamic Time Warping (DTW)-k-Nearest Neighbors (KNN). This method uses the widespread algorithm of nearest neighbours with a distance computed using DTW. Any clustering algorithm must define a distance between two examples. Most of the time, clustering uses Euclidean distance. However, for time series, it is not very appropriate as it ignores the temporal dimension of the data. DTW was constructed to compare time series. A comparison between the two distances can be seen in **??**. DTW assigns a small distance between a series and its accelerated or slowed version. It can compare two series that are of different lengths. For example, it can compare two sequences of people walking even if they don't walk at the same speed.

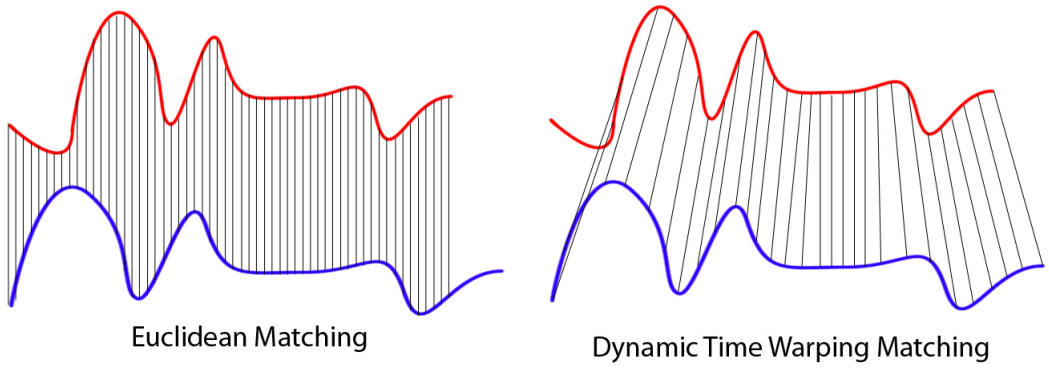

FIGURE 2.2 – A comparison of Euclidean distance and DTW for time series. The two series are similar in shape even if they are not synchronised. DTW assigns a smaller distance than the Euclidean distance. Credits to **wiki3**

Another method based on random forest is the Time Series Forest (TSF) [**tsf**]. Instead of training trees directly on the time series, this algorithm first divides the examples of the dataset into segments of random lengths and random positions. Then, for each segment, a selection of features are computed : the mean, the standard deviation and the slope. The trees are trained using these extracted features.

Another family of approaches that can be used to classify time series is inspired from the dictionary. The main idea of these methods is to pass a sliding window over each series to obtain a constant number of values for each location of the window. Each of these values is transformed into a letter and the concatenation of these letter forms words. The most famous representant of this approaches is the Bag of SFA Symbols (BOSS) method [**Schfer2014TheBI**] that has been declined in several versions such as BOSS ensemble or Contractable-BOSS.

There are many more approaches to classify time series, such as Random Interval Spectral Ensemble (RISE) [**RISE**] or Shapelet Transform Classifier (STC) [**tsc**; **bagnall2020tale**]. The pattern we see emerging with these types of models is intermediate representations. These models take as input the series and output the result. However, in many cases, the series undergoes some kind of transformation and the decision is made based on this transformed input. Some methods have focused on how to best construct the intermediate representation of a series.

2.1.1.5 Time Series Representation

Time series can be transformed in many ways. The transformation can be local (only on some parts of the input) or global (on the whole series). The series can be transformed in another series or in primitive values. For example, the simplest method using primitive consists in representing a series by its mean and variance. A slightly more complex method is to represent it by a histogram of its values [**histogram**].

Frequency analysis is a more complex tool of time series analysis. It can use global or local approaches and results in primitive or time series. It is especially useful when time series are seasonal. Finding the period can make forecasting very easy if the series is strictly seasonal. In addition, if a series is not strictly seasonal, most forecasting models that take seasonality into account will need its seasonality as input.

The most common transformation of series to obtain a frequency representation is the Fourier transform. This approach is based on the assumption that most time series are periodic even if the periodic behaviour is very complex. In any case, a non-periodic function can be considered as a periodic function with an infinite period. Time series are decomposed into a (possibly infinite) weighted sum of trigonometric functions of different periods (also called harmonics). The weight of each harmonic indicates how prevalent a frequency is in the signal. Though not usually used directly for forecasting (except when only relying on seasonality), Fourier transform can be very useful in understanding time series.

Fourier transform only represents the weight of harmonics in the entire time series. There is no way to see that a harmonic is very present in the beginning of a series and absent at the end. The Short-time Fourier transform (STFT) draws inspiration from the Fourier transform. The idea is to divide a series into shorter segments (often overlapping) and apply the Fourier transform to each segment. This reveals the sinusoidal frequency of each local section. The result is a two-dimensional representation (for a univariate time series) with one dimension being the frequency and the other the location (in time) this frequency appeared on. This is often represented as a spectrogram like in **??**. One drawback of this method is that the length of the segments (and their overlapping parts) has a high influence on the result and must be chosen carefully. Just as the Fourier transform, it is not used to perform forecasting directly but it gives a richer representation of a signal, that captures both frequency and location information.

A slightly more complex time-frequency analysis is done via wavelets. Just like STFT, Discrete Wavelet Transform (DWT) performs frequency analysis to capture both frequency and location information. Instead of detecting sinusoidal harmonics, DWT uses different kernels that can detect more complex patterns. To perform wavelet transform, a family of wavelets (based on a kernel) is applied to the series. The result is a unidimensional series that represents the weights of the coefficients of the wavelet family as a function of these coefficients. This new representation can then be used to perform forecasting, often with Box-Jenkins models. A representation of the result of a wavelet transform is shown in **??** It has shown great success in domains such as EEG analysis [**eeg1**; **eeg2**; **eeg3**] where the raw data is extremely hard to interpret but also in other domains such as

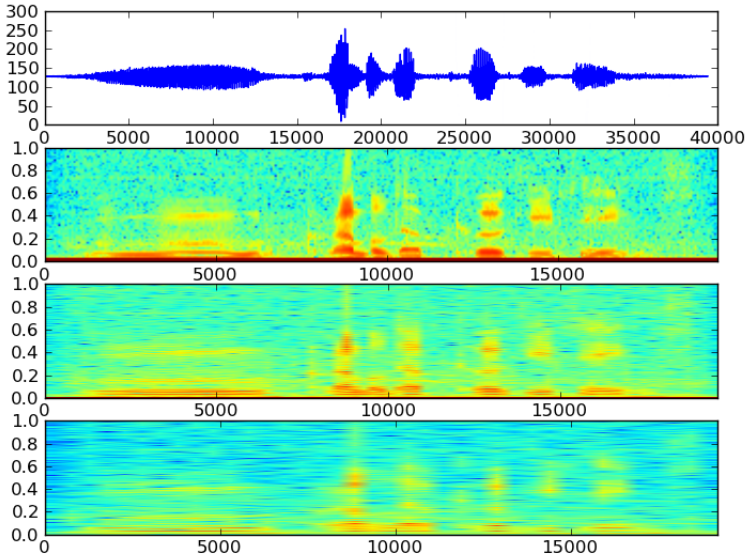

FIGURE 2.3 – A signal and three possible spectrograms of its Short-time Fourier transform (STFT). The only difference between the 3 STFT is the length of the segment respectively 128, 512 and 1024 timestep long. Credits to **wiki2**

car sales prediction [**car_sales**], gas prices forecasting [**gas_price**], solar and wind energy production [**solar**] and many others [**review_wave**].

2.1.2 Deep Learning Approaches

Deep learning approaches aim to learn representations of complex objects, such as time series or images. In a supervised case, the goal of these representations is to obtain better performances on tasks such as classification or forecasting through non-linear features extraction and combination. Algorithms which focus on representation learning as a goal (such as Principal Component Analysis (PCA) and other dimensionality reduction algorithms) that have proved very efficient (even for time series) will be presented in the next section. In this section, we focus on deep learning models that have been used specifically for time series analysis.

Time series forecasting in deep learning is mostly based on two architectures : RNN and CNN. We will first present the RNN and its most common representative : the Long Short-Term Memory (LSTM). As we did not use it in our work, we will succinctly present the CNN before presenting the most recent works that used a combination of both. The goal of these models remains forecasting. However, this forecasting is explicitly performed by learning intermediate representations. They allow us to part from domain knowledge and to learn only from data, without using carefully crafted features designed by experts.

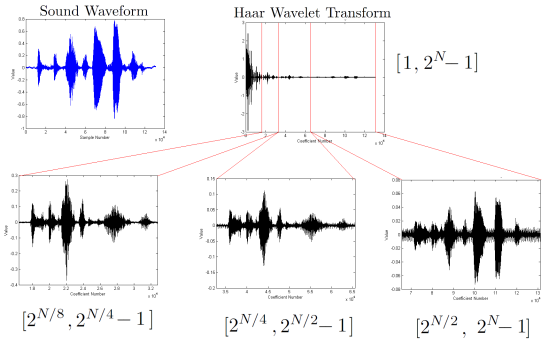

FIGURE 2.4 – "An example of computing the discrete Haar wavelet coefficients for a sound signal of someone saying : 'I Love Wavelets.' The original waveform is shown in blue in the upper left, and the wavelet coefficients are shown in black in the upper right. Along the bottom is shown three zoomed-in regions of the wavelet coefficients for different ranges.'. Credits to wikipedia **wiki2**

2.1.2.1 RNNs

In the recent years, the emergence of RNN has changed many things. Their ability to keep the memory of past trends for a long time makes them extremely powerful in time series forecasting.

Because they only need data and almost no expertise on the nature of the time series, they have become very popular. They allow for the analysis and forecasting of time series with almost no domain knowledge. They learn features on their own, replacing the ones made by experts.

RNNs differs from classical neural networks because they have recurrent connections. For each input data, the networks perform the same function. This function, however, depends not only on the input data but also on the results of the previous computation. Instead of having only one output, RNNs have two. One is the 'classical' output of the neural network, the other is an internal state that can be used as memory and is transferred to the next input. It is worth noting that the two outputs can have the same value. This mechanic allows them to take into account the temporal nature of the data, which classical neural networks cannot do.

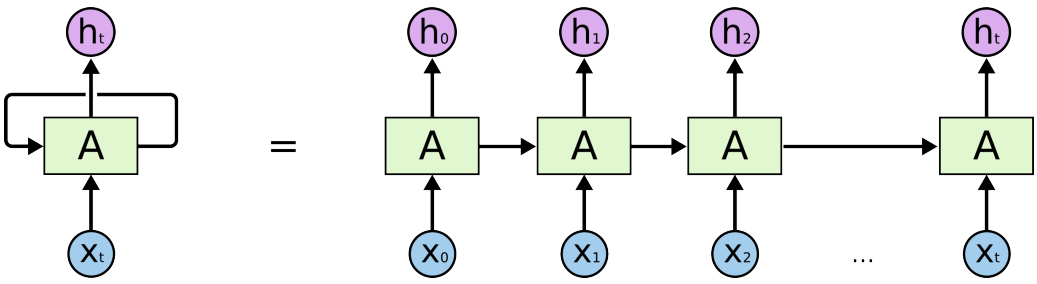

FIGURE 2.5 – A representation of a RNN where A is a chunk of a neural network, x_0^t the input data and h_0^t the output of the network. Credits to **Lstm_blog**

RNN can be used in four distinct settings. The one-to-one, the one-to-many, the many-to-one and the many-to-many. The one-to-one is the classical settings for neural networks, where both inputs and outputs are of the same size (for classification of fixed size input). The one-to-many has inputs of fixed size but outputs of varying size. The many-to-one has inputs of varying size but outputs of fixed size (classification of sentences for example). Finally, many-to-many has inputs and outputs of varying sizes. This last mode can be split into two modes depending on the synchronisation. Indeed, there can be a shift between inputs and outputs if, for example, the whole input needs to be read before the outputs can be generated (in translation for example). The **??** illustrates the different modes. The most interesting mode for our work is the many-to-one. It allows representing sequences of varying length into an embedding of fixed dimensions. This embedding can then be used in other models that cannot handle inputs of varying sizes.

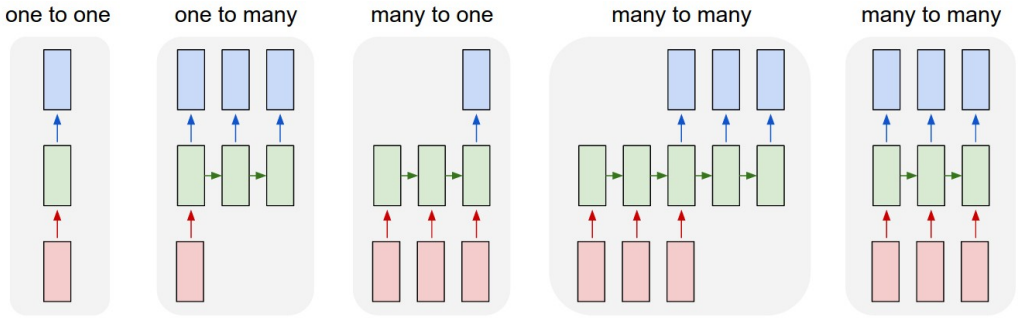

FIGURE 2.6 – "Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right : (1) Simplest mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). (2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words). (3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). (4) Sequence input and sequence output (e.g. Machine Translation : an RNN reads a sentence in English and then outputs a sentence in French). (5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case there are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.' Credits to **rnn_effectiveness**

More formally, RNN cells have a hidden state, usually noted $\boldsymbol{h}$. A cell is comprised of two weight matrices, W_h and W_{in} and their associated bias ($\boldsymbol{b}_{in}$ and $\boldsymbol{b}_h$). At each time step, it receives the previous hidden state $\boldsymbol{h}_{t-1}$ and the input $\mathbf{x}_t$. The output of the cell is also its updated hidden state :

$$\boldsymbol{h}_t = \tanh(W_{in}\mathbf{x}_t + \boldsymbol{b}_{in} + W_h\boldsymbol{h}_{t-1} + \boldsymbol{b}_h) \tag{2.4}$$

Because of their recurrent nature, RNNs are trained using Backpropagation through time (BPTT). This technique unfolds the RNN to compute the gradient, taking into consideration

that the same parameters (W_{in}, $\boldsymbol{b}_{in}$, W_h and $\boldsymbol{b}_h$) are used for each time step. This algorithm, however, can sometimes lead to very inefficient learning, due to an issue called gradient vanishing.

Gradient vanishing occurs when the BPTT makes the gradient progressively smaller as it goes back in time. This means that the earlier input computations will not meaningfully impact the parameters update. This issue arises because the derivative of $\tanh$ is smaller than 1 and the chaining rule will make this decreasing effect compound.

Though less common, gradient exploding is worth mentioning. It is the exact opposite of the previous issue. Instead of becoming too small, gradients become too large. This issue arises when the norm of the weight matrices are too large. It is less common than vanishing gradients and it is also easier to fix and diagnose, and therefore often not considered as important an issue.

Another limitation of the classical RNN (partially due to the vanishing gradient) is that although they can theoretically keep memory of past input for as long as needed, in practice, when the output must depend on an older, distant input, RNNs tend to have forgotten the needed information. As the gap between the time step of the input and the time step where it is needed grows, the RNN becomes unable to connect the information. This connection between older inputs to later outputs is called long-term dependency.

It is to alleviate these two problems of RNNs (long-term dependency and gradient vanishing) that LSTMs were created by **LSTM**. LSTMs are a gated version of RNNs that allows for the selection of what information to keep or forget from memory and input. This allows them to remember information for a very long time. The difference with classical RNNs is that instead of performing a simple matrix multiplication followed by activation (one layer of neural networks) their internal behaviour are much more complex and contains four internal layers of neural networks. The internal state of a LSTM is also not the same as its output. Some information is kept internally for the next step but not outputed. **??** shows a representation of a cell of a LSTM.

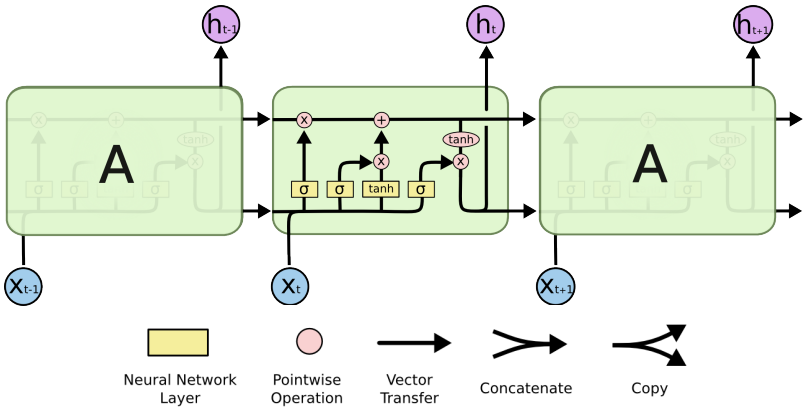

FIGURE 2.7 – A Representation of a Cell of LSTM Credits to **Lstm_blog**

To go into details, the **??** shows the gates that compose the LSTM. The forget gate chooses what to keep from the cell state by looking at the hidden state and the input. The

input gate decides what to keep from the input. Combined with a tanh layer, it creates a new candidate value for the cell state. The input and forget gates outputs are then used to update the cell state (by forgetting some information through the forget gate and adding information from the input transform by the input gate). Finally, the output gate uses this new cell state in combination with the input to create the output of the cell.

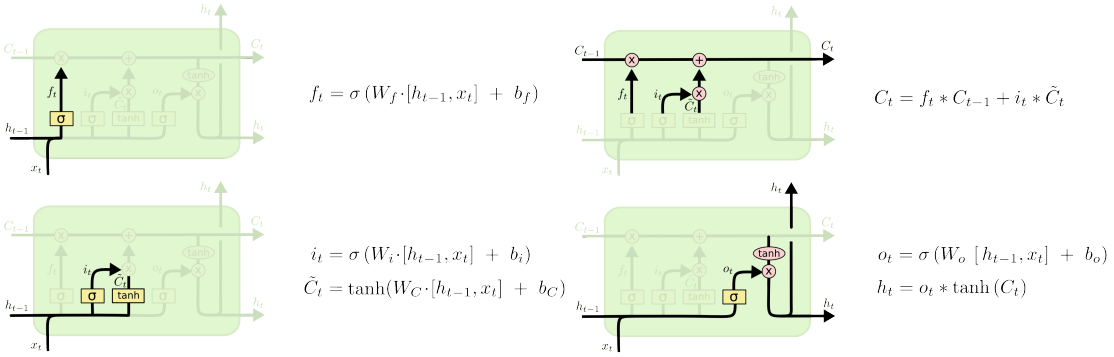

FIGURE 2.8 – The gates of an LSTM cell. The forget gate (upper left). The input gate (bottom left). The output gate (bottom right) and the cell state update (upper right). Credits to **Lstm_blog**

This solves the gradient vanishing problem because the BPTT is done on four different parts of the cell. Specifically, the forget gate's vector, combined with the addition of four different terms, make it very unlikely that the gradient would tend toward 0. It therefore reduces the probability of vanishing gradient. This mechanism allows LSTM to keep past events in memory for many time steps.

LSTM is one of the most widely used recurrent model [**review_lstm**]. Its applications range from time series modelling to Natural Language Processing (NLP).

We will focus here on their application in the scope of time series modelling, with some impressive results achieved on classification and forecasting tasks.

LSTMs have been used to classify time series [**review_lstm**]. One such common application is the prediction of faults. In **turbine**, the authors specifically used an LSTM-based framework in order to avoid reliance on experts. They outperformed the traditional methods of fault classification in wind turbines by using only raw data time series coming from sensors. A study of how to design flexible fault diagnosis models using LSTM was published by **faultdia**, with a case study on nuclear energy.

Unlike the historical approaches, LSTMs make no assumptions about the stationarity, the linear dependence or the sequence correlation. This made them very useful in many forecasting tasks where they vastly outperformed traditional benchmarks such as random forest or standard logistic regression [**review_lstm**]. They have proved superior to traditional baselines in various applications such as building energy usage prediction [**building**], carbon emission levels [**carbon**], health predictive analytics [**healthpredictive**] and many other [**review_lstm**]. LSTMs can also be used with input data of various natures. For example, in **taxiNY**, they combined time series observation with text input in order

to predict taxi demand in New York. The method used there can easily be adapted to all sorts of applications that need handling of heterogenous data.

Our goal remains to learn representations. As it is difficult to supervise directly, the use of the forecasting task allows us to have a natural supervision. Indeed, the sequential nature of the data and the model makes the prediction of the next values the most intuitive task. Thanks to their hidden content, LSTM (and RNNs in general) learn representations of the time series. At any given time step, the hidden layer of the network contains information of the input time series thus far.

2.1.2.2 Hybridisation of CNNs and RNNs

Another architecture that can be used for time series forecasting is CNN. CNNs are neural networks that alternate layers of convolutions and layer of pooling and often end with classic fully connected layers. **??** shows a CNN built to perform image classification.

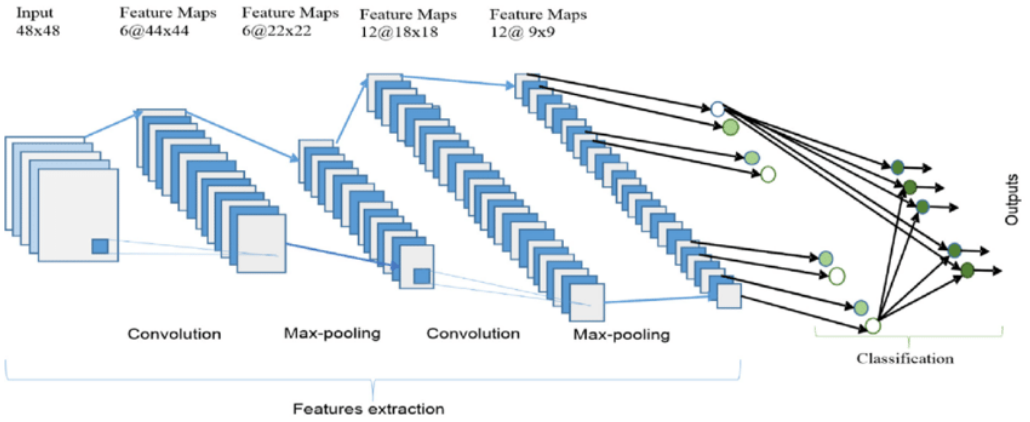

FIGURE 2.9 – An illustration of a CNN with 2 layers of convolution, one fully connected layer and one classification layer. Credits to **convnet**

A layer of convolution is comprised of a number of filters. Each filter is convoluted with the input, the resulting matrices (for 2D convolution) are called the feature maps. The nature of the convolution operation depends on the model. There are several hyper-parameters in these models, such as stride, padding or size and number of filters. A few possibilities for these hyperparameters are shown in **??**.

For the case with no padding and no stride, if we denote X the input matrix (in two dimensions) of a convolutional layer and W_i a filter of this layer, the resulting feature map F_i is computed through :

$$F_i = X \star W_i \tag{2.5}$$

where $\star$ is the convolutional operation. This operation is done for each filter W_i in the convolutional layer resulting in as many features maps as they are filters.

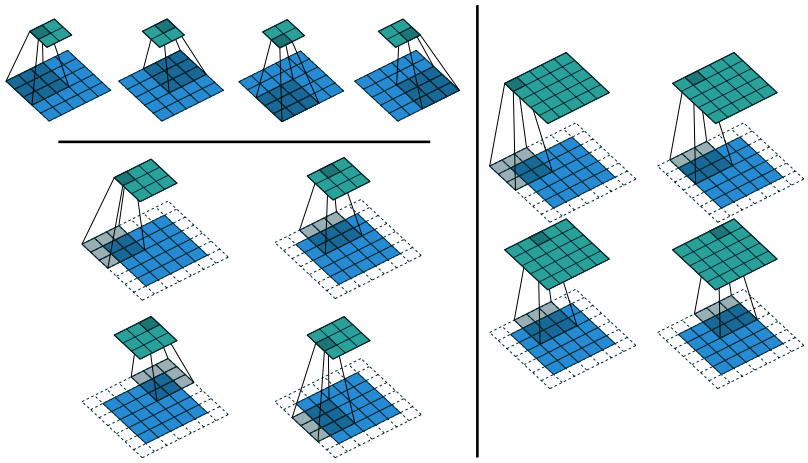

FIGURE 2.10 – An illustration of the process of convolution by filter of size 3×3 with different hyperparameters. The upper left use no padding and a stride of 2. The lower left use a stride of 2 and a padding of 1. The right use no stride and a padding of 1. Credit to **dumoulin2016guide**

These features maps are then passed through a non-linear activation function. The result is the input of the pooling layer. The pooling layer subsample its input to reduce its size. Each feature maps is divided into neighbourhoods. For each neighbourhood, only one value is computed. The value is usually computed either as the max value or the averages of all the values of the neighbourhood. The polling process is illustrated in **??**.

If we denote F the input feature map (in two dimensions) of a pooling layer and P_i the neighbourhood that compose it, the resulting matrix X is computed through :

$$X_i = pool(P_i) \tag{2.6}$$

where X_i denote the i^{th} component of X and $pool$ the pool function used. Usually $pool$ is either the $\max$ function or the mean function.

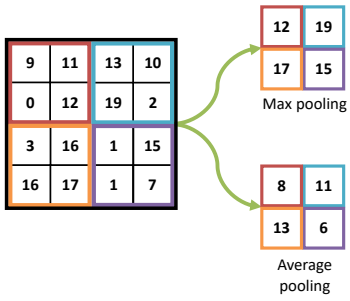

FIGURE 2.11 – An illustration of the process of max and average pooling on a 4×4 feature map with 2×2 neighbourhoods.

For the task of forecasting, the traditional CNN is slightly modified so that there can be no leakage from the future to the past. The classical convolution would allow using some time steps in the future to predict the present. In computer vision this behaviour is

desirable because pixels are not ordered but in time series, it should not happen. CNNs also needs to be able to handle input of varying size to model time series. This is done via the use of 1D-Fully Convolutional Network (FCN) with zero padding at each layer and causal convolution that ensures that output at time t is only dependant on inputs of time t or earlier. To tackle long-term dependencies, dilated convolution can also be used. Dilated convolution allows spreading the filter on a larger number of inputs without increasing their size. An example of diluted convolution is shown in **??**.

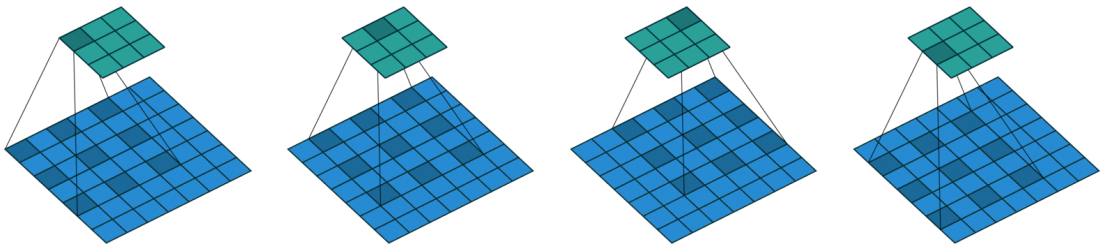

FIGURE 2.12 – An example of dilated convolution with a filter 3×3 and dilation of 1. No striding or padding. Credit to **dumoulin2016guide**

This type of convolution allows looking back further in the past than the kernel size. A comparative of recurrent and convolutional network on different tasks related to time series modelling can be found in [**bai2018empirical**]. Though this study seems to indicate that, in many cases, CNN achieves better performances, the hybridisation of the architectures that retain the advantages of both seems to be the best solution.

One example of such hybridisation is the Convolutional LSTM [**convlstm**]. In this model, the four fully connected layers that composed the LSTM are simply replaced by convolutional layers. The architecture is illustrated in **??**. Convolutional LSTM have been used in a variety of applications with great success, like precipitation nowcasting [**convlstm**] or pandemic spread prediction [**Paul2020AMS**].

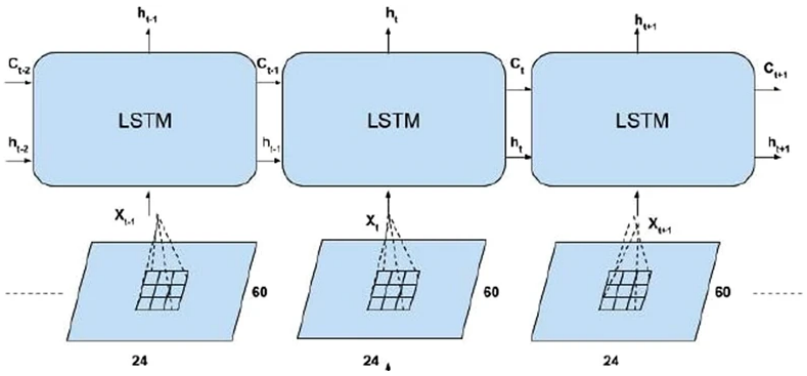

FIGURE 2.13 – An illustration of the hybridisation of LSTM and CNN. Credits to **convLSTM_fig**

Different hybridisation of CNNs and RNNs have been researched, especially in the domain of weather recognition and forecasting where the alliance of the spatial capability

of CNN and the temporal capabilities of LSTM are a very desirable trait [**CHEN2019783**; **Zhao2018ACA**].

In the very different domain of real-time parking occupancy, **parking** have shown that the concurrent use of RNN and Graph Convolutional Neural Network (GCNN) outperforms the other approaches, including the one based on classical LSTM.

This hybridisation combines the temporal power of the LSTM with the representation learning power of the CNN. This ensures that the hidden content of the RNN is a very good representation of the input.

2.1.3 Conclusion

Time series modelling is a long-standing research problem with applications in many different domains. Forecasting is one example of modelling and, for us, it provides supervision for learning representation. AR models are one example of classical forecasting models. Their drawback is that they cannot be used for data that is not linear or stationary. Tree-based models, on the other hand, can be used even on non-linear data to perform either classification or forecasting. However, even with these models, a better representation of the data as input can improve performance. Historically, this transformation required expert domain knowledge to be effective. Recently, data driven approaches have appeared. Models such as RNN and CNN transform data to acquire a better representation of it, in order to perform certain tasks like classification or forecasting. These intermediate representations are of interest to us, even though they are often incomplete. Indeed, they are highly skewed toward the task at hand (e.g. forecasting or classification) which means it can lack some information about the input that is not needed to perform this specific task. The advantage of representations that are not skewed toward the accomplishment of a specific task is re-use. If we can obtain a unique representation of a complex object that contains all the information of the original object, this representation can be used in a variety of models to perform a lot of different tasks.

2.2 Representation Learning as a Goal

Representation learning focuses on learning representation of complex objects. These representations can then be used to perform many tasks such as classification or generation. As explained above, though CNN, RNN and their hybridations do perform representation learning, the use cases presented always focused on a task (classification, forecasting...). Therefore, they learn incomplete representation of the input object, skewed toward the task at hand. To be able to re-use them, we want our representations to be free of tasks. In NLP for example, the crucial part is to understand the sequential shape of the raw data : the structure of the sentences. Transforming text data into objects that can be handled by models is a huge challenge. Even though progress has been made with *word2vec* [**word2vec**] or *fasttext* [**fasttext**], these are focused on the word level. Once a representation for each word is created, the issue of how to aggregate these representations to

represent entire sentences appears. And then how to aggregate these representations of sentences into representation of documents. Representations of complex objects cannot easily be combined to create representations of more complex objects. These issues of the level of representation and their aggregation are found in many domains. For example, in computer vision, can we represent a video with a sum of the representations of its frames.

The issue of representation learning did not start with deep learning, though it is at the heart of these approaches. This first attempt at simplifying representation was by dimensionality reduction. In deep learning, the purest form of representation learning is through generation. Indeed, being able to generate data corresponding to an input dataset means that our model has completely understood the data.

2.2.1 Dimensionality Reduction

When dealing with complex objects, the first intuition is often to find a way to simplify them. This can be done for several purposes. The most current and oldest one is for representation. We can sometimes understand many things about our data simply by representing it in 2D or 3D. Because visual reasoning can be extremely fast, seeing all of it in simple representation can be very helpful. Another common purpose of simplifying data is for regularisation. To ensure that only important data is encoded, reducing its size is a solution. It can be useful to avoid overfitting but also to minimise noise in the data. Often, this will imply finding smaller representations. This is what led to the many existing methods of dimensionality reduction. With large amounts of complex data, some of which can be extremely correlated, the aim is to 'sum up' the data into a smaller representation that contains as much information as the original one. As this is often impossible, most dimensionality reduction will have some loss of information that should be minimised.

2.2.1.1 PCA

One of the oldest and most widely used methods of dimensionality reduction is the PCA [**PCA**]. Its goal is to reduce dimensionality while minimising variability. This is done by creating new uncorrelated features that can best represent the dataset. The PCA maximises the amount of variance kept by using these features to represent the dataset. Finding these principal components is akin to solving an eigenvalue/eigenvector of the dataset. As this method is widely known, we will not go into details here.

2.2.1.2 Autoencoder

Autoencoders have been popular in the field for decades, and applications were present in the '80s [**Bourlard2004AutoassociationBM**; **GoodfellowDL**]. A representation of an Autoencoder (AE) can be found in **??**.

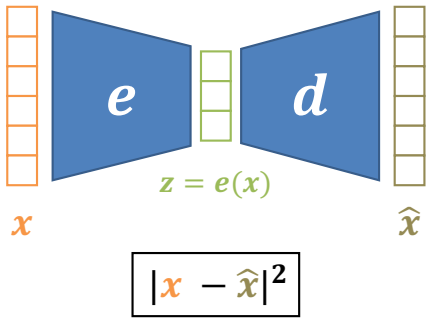

FIGURE 2.14 – An autoencoder using a L2 loss. The input $\mathbf{x}$ is given to the encoder e that outputs the code vector $\boldsymbol{z}$. This code is given to the decoder d that attempts to reconstruct the input.

The general idea of AE is quite simple. In order to learn a representation of an object, you must be able to reconstruct the object from its representation. An AE is comprised of an encoder e and a decoder d. The encoder learns the salient features of the input $\mathbf{x}$ and encodes them into a code vector $\boldsymbol{z}$, also called latent vector. This vector is then given as input to the decoder d that must reconstruct the original input and produces $\hat{\mathbf{x}}$. Both parts are trained jointly, with one goal, for the input of the encoder and the output of the decoder to be identical. Formally :

$$\boldsymbol{z} = e(\mathbf{x}) \tag{2.7}$$
$$\hat{\mathbf{x}} = d(\boldsymbol{z}) \tag{2.8}$$
$$L(\mathbf{x}, \hat{\mathbf{x}}) \tag{2.9}$$

where L is a function measuring the similarity between $\mathbf{x}$ $d(e(\mathbf{x}))$ such as the Mean-Squared Error (MSE).

If the dimension of the latent vector is smaller than the input, then this is performing dimensionality reduction. In any case, the latent vector is a new representation of the input and the encoder should be learning its most salient features. If the decoder is a linear function and the loss used the MSE, then the AE is very similar to the PCA. Using non-linear decoder (such as neural networks) leads to a kind of generalisation of the PCA [**GoodfellowDL**].

A very important thing to note is that if $\boldsymbol{z}$ has a dimension too large or that e and d have a too large capacity, they may not learn anything. In the first case, if $\boldsymbol{z}$ has the same dimension as $\mathbf{x}$, it is obvious that e and d can learn to be the identity function and therefore not learn anything. For the second case, an extreme example in which $\boldsymbol{z}$ if of size 1 but d and e have large capacity, then e and d could learn a mapping between each example of the dataset and its index. Neither of this scenario is likely to happen as they are extreme. However, we could imagine approaching such scenario in real use cases and come close to these possible failure cases of AEs.

Besides its usefulness for dimensionality reduction and features learning, a number of variations of AEs were created to fulfil many purposes. In many cases, by just adding a regularisation to the vanilla AE, preventing it from learning a perfect reconstruction has proved to be very useful. These models are called regularised AEs.

One such example is the Denoising AE [**DAE**]. In this case, the input given to the encoder is $\tilde{\mathbf{x}}$ a copy of $\mathbf{x}$ that has been modified by noise. Because the output is still compared to $\mathbf{x}$ (and not $\tilde{\mathbf{x}}$), the model must learn to remove this noise.

Sparse AEs, presented by **SparseAE2007**, are trained to learn sparse representations (where there are fewer coefficients of z that are non-zero) by simply adding a function dependant on z, called the sparsity penalty, to the loss function. There are several possibilities for the sparsity penalty, leading to different results [**SparseAE2008** ; **ZENG2018643**].

Sequential AEs are regular AEs where the encoder and decoder are made of RNN, often LSTM. They allow for the learning of representations of sequential data, including time series. In this case, the encoder is used in the many-to-one mode of RNNs where the input can be of varying size but the output is of fixed size. The output is our latent vector. The decoder is used in the one-to-many mode and takes as input the latent vector and the output can be of varying length. As explained above (see **??**), this is particularly interesting in our work because it transforms sequences of different sizes into representations of fixed size. These representations can then be used for other tasks where models cannot handle inputs of varying length.

One limitation of the AE is that the space of representation has no reason to be regular of well organised.

What we mean by a regular or organised latent space is covered by two concepts : **continuity** and **completeness**. Continuity is the fact that two points that are close in the latent space should not give two completely different decoded representations. Completeness is the fact that a point sampled from the latent space (according to the distribution of the encoding of the examples) should decode into a 'real' example. For example, it means that interpolating between the representation of two examples should give a believable example once decoded.

This limitation of AE is well documented in the literature and illustrated in **??**.

Models that do have regular latent space that can be sampled to create new data are generative models. AEs are similar to generative models because they have the same sort of 'unsupervised supervision'. The only available supervision is the reconstruction of the original examples. The difference becomes obvious when trying to transform the AE into a generative model by using just the decoder and sampling the latent space. The generated examples are, in their overwhelming majority, nothing like the original examples. The main difference between AEs and generative models is the structure of their latent spaces.

To generate new examples from a latent space, there must be some sort of continuity and completeness in the space. In turn, this means that the learned representations are more structured and meaningful. For example, a distance between representation should make sense : close representations should represent similar examples, while distant representations should represent different examples.

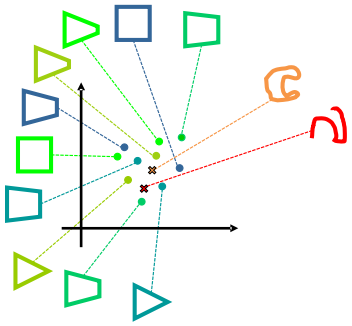

FIGURE 2.15 – A representation of the latent space of a classic autoencoder. The dots represent actual examples from the dataset, while the crosses represent the reconstruction of sampled points in the latent space. In this case, the space is highly irregular, meaning that two points that are close in the latent space do not necessarily decode into close examples and a point sampled in the latent space close to 'real' latent vector will not decode into an example that could belong to our dataset.

2.2.2 Deep Generative Model

This work focuses on deep generative models not based on graphical models. Deep graphical models such as deep Boltzmann Machine are not discussed nor are Bayesian models. Markov models have been used to generate sequences (and especially sentences) but they are not considered here. They often require simplifications (because of the first order dependency) that makes them less efficient.

In addition to their structured latent spaces, deep generative models have the advantage of ensuring a complete understanding of the dataset. Indeed, to be able to generate examples that follow the distribution of the original dataset, the model must learn this distribution. By construction, generative models are models that provide a way to generate new data points that belong to the same distribution as the original dataset.

The conjunction of structured latent space and generative ability is very interesting for us. Generative models have tremendous predictive power. They allow for prediction from no data (if we exclude the training data). This can be seen as forecasting at infinite time horizon. Conditional generative models can generate while respecting a condition. If this condition is sufficiently meaningful, then we are truly forecasting from only context data. As these models are mostly based on classical generative models, they share with them notations and frameworks. We will first present the classical framework of the generation task and present their conditional counterparts in **??**.

In this framework, we denote the original dataset we want to reproduce J_x. This dataset contains N examples. We denote $\mathbf{x}$ one example of this dataset. The generation task assumes that there is an unknown underlying distribution, we denote $p(\mathbf{x})$ from which the examples have been sampled to form J_x. Generative models aim at creating new realisations of the underlying distribution.

The generation task is often classified into the unsupervised learning category. Depending on how it is tackled, there may be no need for labels or annotation of the dataset. To generate new data that match the underlying distribution, the model has to learn this distribution and totally understands it. This ensures that the learned representation contains a maximum of information about the example. This representation can then be used to perform other tasks. For example, in NLP, because any other task is incredibly difficult to supervise, the learning step is done via generation, in order to extract the sense [**bert**]. This ensures that the model understands the language model, that is, the distribution probability of sequences of words in a language. This language model is only language dependent and does not depend on the task. Once learned, it can be transferred to perform other NLP tasks. These tasks can be as diverse as sentiment analysis, summarisation or question answering. The language model is often learned by learning representation of individual words in a latent space. Each word is mapped to an embedding. However, the same word can have different meaning depending on context. Contextualised embeddings were introduced so that the representation of a word would depend on the other words of the sequence.

The generation task has been tackled before deep learning in several ways, using graphical models or Gaussian mixture models. However, because high dimensionality data requires complex relations, it has not been very successful. The first attempt to use deep learning was the Generalized Denoising AE, presented by **pmlr-v32-bengio14** and **bengio2013** where they described a Monte-Carlo Markov chain that is parametrised by neural networks and therefore trainable through classical back propagation.

More recent approaches have slightly modified the task. Instead of directly learning $p(\mathbf{x})$, they try to map to it a chosen latent distribution $p(\boldsymbol{z})$. The challenge then becomes to learn a parametrised function G that maps $p(\boldsymbol{z})$ and $p(\mathbf{x})$ so that $p(G(\boldsymbol{z}))$ approximate $p(\mathbf{x})$. Generating new point is done by sampling $p(\boldsymbol{z})$ (which is generally a simple distribution such as $\mathcal{N}(0, I)$) and applying G to the sample. The difficulty is to train the function G.

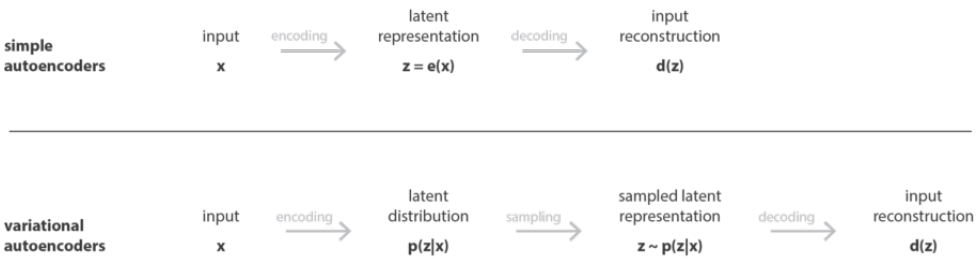

FIGURE 2.16 – Difference between classical autoencoder (deterministic) and variational autoencoder (probabilistic)/ Credits to **vae_blog**

There are two main families of models based on this approach, the Variational Auto-Encoder (VAE) and the Generative Adversarial Network (GAN), which we will now present succinctly. A new approach, popularised by **dinh2014nice** and **rezende2015variational**,

the Normalising Flow will not be discussed. To describe it briefly, the idea is to learn a chain of invertible functions that directly maps $\boldsymbol{z}$ to $\mathbf{x}$. This chain can easily be learned through the maximisation of the log-likelihood (which can be evaluated exactly). Because the transformation is invertible, the exact posterior is known and can be used for inference. However, this reversibility also means that the latent space must have the same size as the space of observation, which makes them less relevant to this work.

We focus on the image generation task, as it is the most documented and successful application of deep generative models.

2.2.2.1 VAE

VAEs are AEs that are trained and regularised to ensure that their latent space of representation is structured (continuous and complete). This is what allows them to perform generation.

VAEs were presented by **autoencoding** and **rezende2014VAE**. The difference with traditional AE is that instead of encoding an input into one latent vector, VAEs encodes the input into a distribution over the latent space. This distribution is then sampled and this sample is given as input to the decoder. This forces the latent space to be complete and continuous.

To parallel with AEs, in this probabilistic model, our encoder and decoder are now described by two distributions. The decoder is noted $p_\theta(\mathbf{x}|\boldsymbol{z})$ because it represents the distribution of the decoded variable given the latent factor. It is parametrised by a set of parameters θ. The encoder is noted $q_\phi(\boldsymbol{z}|\mathbf{x})$ because it represents the distribution of the latent factor given the decoded one. It is parametrised by a set of parameters ϕ. The VAE mixes a stochastic encoder (the inference model) with a probabilistic decoder (the generative model).

VAEs were not designed from AEs but rather came from a variational inference framework. Their main purpose is to be generative models. Therefore their formulation is based on notation and ideas coming from probabilistic models and variational inference. The idea of this framework is to set a parametrised family of distributions (for example the family of Gaussians, whose parameters are the mean and the covariance) and to search for the best approximation of our target distribution amongst this family.

From a dataset of observations J_x, VAEs aim at reproducing the distribution $p(\mathbf{x})$ of the dataset. Formally, it means minimising the Kullback–Leibler divergence (KL) between the distribution induced by our model $p_\theta(\mathbf{x})$ and $p(\mathbf{x})$. It can be shown that this is equivalent to maximising the expectation of the log-likelihood of $p_\theta(\mathbf{x})$. We can show that :

$$\log\ p_\theta(x) = KL(q_\phi(\boldsymbol{z}|\mathbf{x})||p_\theta(\boldsymbol{z}|\mathbf{x})) + \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\boldsymbol{z})] - KL(q_\phi(\boldsymbol{z}|\mathbf{x})||p(\boldsymbol{z})) \quad (2.10)$$

where $q_\phi(z|x)$ is an approximate of $p_\theta(z|x)$ that will be learned via variational inference.

Because $p_\theta(\boldsymbol{z}|\mathbf{x})$ is unknown, we cannot optimise this loss directly. However, as the KL is positive, we can instead optimise the Evidence Lower BOund, noted ELBO :

$$\log\ p_\theta(\mathbf{x}) \geq \mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - KL(q_\phi(\boldsymbol{z}|\mathbf{x})||p(\boldsymbol{z})) \tag{2.11}$$

To continue our parallel with AEs, the first term of the Evidence Lower Bound (ELBO) is a reconstruction loss. We want to maximise the probability that our decoder outputs a sample close to the original input $\mathbf{x}$. The second term is the regularisation, where we want our encoder to encode observation into a specific distribution $p(\boldsymbol{z})$ that will force our latent space to be complete and continuous. This model is illustrated in **??**.

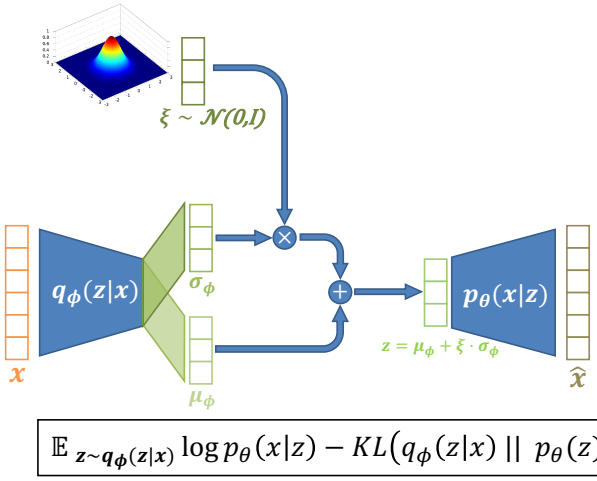

FIGURE 2.17 – A Representation of a Variational Autoencoder with the re-parameterisation trick. The encoder outputs the parameters of the distribution we want our latent space to fit (here a Gaussian). To be able to perform back propagation, another distribution is sampled and the sample is transformed using the outputed parameters. The transformed sample is given to the decoder to reconstruct the original example. Once trained, to perform generation, the desired distribution is sampled and the sample is given to the decoder, which outputs a new example.

During training, to compute the ELBO, it is necessary to evaluate the log-likelihood $p_\theta(\mathbf{x}|\boldsymbol{z})$. This requires to choose a family of distribution for $p_\theta(\mathbf{x}|\boldsymbol{z})$. Because the KL between two (possibly multivariate) Gaussian distributions is known in a closed form, this is often the path taken. Whichever family is chosen, some practical considerations must be taken into account so that the KL can be computed.

To compute the ELBO, it is also necessary to compute the stochastic gradient estimation of $q_\phi(\boldsymbol{z}|\mathbf{x})$. The issue is that the gradient cannot be back propagated through the sampling step (the encoder returns a distribution and not just a latent vector and sampling $q_\phi(\boldsymbol{z}|\mathbf{x})$ is needed to get an input for the decoder). The Reparametrization Trick was presented to resolve this issue. Now wildly used, this trick is quite simple : instead of sampling $\boldsymbol{z}$ directly ($\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\mathbf{x})$), we use an auxiliary independent variable $\zeta \sim p(\zeta)$. Because $q_\phi(\boldsymbol{z}|\mathbf{x}) = \mathcal{N}(\mu_\phi, \sigma_\phi)$, we can easily reparametrized with :

$$\boldsymbol{z} = \mu_\phi + \sigma_\phi \cdot \zeta, \quad \text{with} \quad p(\zeta) = \mathcal{N}(0, I) \tag{2.12}$$

The gradient can then be back propagated through the encoder that is deterministic. It now outputs the variables that parametrise the distribution and not the distribution itself.

The constraints on the latent space of the VAE forces the space to be much more regular as in a classical AE. Using the same conventions as in **??**, **??** could represent the latent space of a VAE

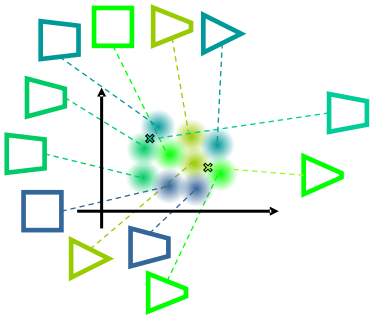

FIGURE 2.18 – A representation of a latent space of a VAE. The space is more organised that in classical AE. The blurred disks represent the distribution that is constructed by the encoder. Points sampled in one distribution should decode into similar examples. Interpolation in the latent space should decode into slowly changing examples

2.2.2.2 GAN

Presented by [**Goodfellow2014GenerativeAN**], GANs try to leverage the discriminative power of neural networks to improve their performances in generation. They are based on an adversarial setting in which two networks are trying to optimise the same loss, only one is trying to maximise it, while the other is trying to minimise it. Specifically, there is one network, the generator, noted G, that learn a mapping between a noise distribution $p(\boldsymbol{z})$ and the distribution of observations $p(\mathbf{x})$. The other network, called the discriminator and noted D, is a classifier whose goal is to distinguish between a sample from the real dataset and a sample generated by G. D is therefore a binary classifier that distinguished between real and fake examples coming from G. G, on the other hand, has the opposite objective of creating samples that D cannot distinguished from the real observations. An illustration of the model can be found in **??**.

This leads to a minimax formulation :

$$\min_G \max_D V(D,G) = \min_G \max_D [\, \mathbb{E}_{\mathbf{x}\sim p_\mathbf{x}}[\log\ D(\mathbf{x})] + \ \mathbb{E}_{\boldsymbol{z}\sim p_\boldsymbol{z}}[\log\,(1 - D(G(\mathbf{z})))]]$$

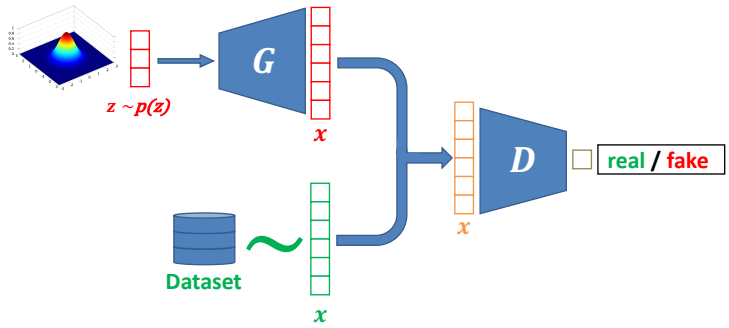

FIGURE 2.19 – A representation of a Generative Adversarial Network with a Gaussian noise distribution. The goal of the discriminator D is to learn to distinguish between generated samples and real examples. The goal of the generator G is to transform samples from a Gaussian distribution into examples that D cannot distinguish from the data from the real dataset. After training, the Gaussian is sampled and the sample given to G. The output of G is a new example.

It was shown that, if D is optimal, this adversarial objective minimises the Jensen-Shannon Divergence between $p(\mathbf{x})$ and $G(p(\mathbf{z}))$ and therefore that both distributions are similar.

In the original paper, the authors disclosed that this minimax adversarial technique was unstable and even often failed to learn anything. Since then, many improvements have been made and many variations around GANs have been designed. As this is not our main focus, we will only briefly cite the most important ones.

The first amelioration was presented by the same authors [**Goodfellow2014GenerativeAN**]. They noted that a common cause for failure was the difference in difficulty in the two objectives (D's objective is easier that G's) which lead D to become 'too good too fast' and essentially zeroing out the gradient passed onto G. They slightly modified the objective of G so that instead of minimising the probability that D classifies $G(z)$ as fake, it maximises the probability that D classifies it as true. This allows for G to get stronger gradient.

A class of GAN that was a great advance are the f-GANs. f-divergences are a family of functions that measure distances between probability distributions. **nowozin2016fgan** showed how modifying the objective function of GANs could lead to the minimisation of different f-divergences. This family of GAN is almost never used 'as is' today but it was the basis of many important improvements of GANs.

Maybe the most notable version of GAN, the Wasserstein GAN (presented by **arjovsky17a**) built on the f-GAN to show that instead of minimising an f-divergence, using the Wasserstein-1 distance would improve stability and performances. Although the first formulation was unstable due to the necessary addition of constraints (especially a Lipschitz constraint on D), the latter improvements for instance by **GulrajaniAADC17** and **abs-1812-04948** made it a must-see of the GAN galaxy.

Developed almost simultaneously by two sets of authors : **DonahueKD17** and **dumoulin2017adversarial** Bi-GAN (bidirectional GAN) adds another function that learns the reverse mapping from

$\mathbf{z}$ to $\mathbf{x}$. This third part of the model, noted E is trained jointly with D and G and like G models $p(\mathbf{x}|\mathbf{z})$, E is trained to model $p(\mathbf{z}|\mathbf{x})$.

2.2.2.3 Conclusion

These two families of models have achieved great results. To simplify, VAEs must find balance between the two terms of the ELBO. One ($p_\theta(\mathbf{x}|\mathbf{z})$) tends to lead to good reconstruction, while the other one (the KL) tends to regularise and simplify the latent space. The drawback of this formulation is that VAEs often produce less realistic samples than other deep generative models. In image generation for example, VAEs are known to produce blurry images. They are therefore scarcely used when high-frequency features are needed. Some work has been done to overcome this weakness. However, this often leads to computer-intensive methods. On the other hand, VAEs have also shown a great potential of interpretability. When the chosen prior is a diagonal Gaussian, it has been observed that the KL term leads to sparser and more disentangled code. This is why VAE are often used in domains where good latent spaces are needed.

GANs on the other side have the benefit of not using likelihood. They can work with implicit distributions and are therefore easy to interface with other deep models. Their flexibility has made them very popular and they have been used in a great variety of tasks. Their disadvantage is their instability. GANs are hard to train and very prone to mode collapse and mode dropping. They usually fail to capture points that don't correspond to important modes ; which can be seen as underfitting.

These techniques of generations allow for a continuous and complete latent space. However, we still have no control over what each component of the latent space stands for. In a way, there are still very much black boxes. What we want to do is open this box, making the model explainable. One way to do this is to match components of the latent space to meaningful concepts in the real world. Both VAEs and GANs were improved and modified to allow conditional generation. These variations of the model often allow for disentanglement. They can generate examples according to conditions that are linked to the real world.

2.3 Conditional Generation and Disentanglement

Though disentanglement has been the focus of a lot of papers in the last few years, it appears there is not much consensus on one definition of it or one way to measure it [**higgins2018definition** ; **kumar2018variational** ; **mathieu2018disentangling**]. The only consensus is that a disentangled representation maps latent factors to a *generative factor*. The problems arise when we must define generative factors. Outside of toy datasets where we have clearly label ground truth for the generative process, this is not clear.

The main idea, however, remains clear. We want a latent space (or latent spaces) that are structured in relation to the real world. We want to have some interpretability for the latent representations.

Taking the same notation as in :fig :latent$_A E@cref$

:fig :latent$_A E@cref$

:fig :latent$_A E@cref$

:fig :latent$_V AE@cref$

:fig :latent$_A E@cref$

:fig :latent$_A E@cref$

:fig :latent$_V AE@cref$

:fig :latent$_V AE@cref$

:fig :latent$_A E@cref$

:fig :latent$_V AE@cref$

:fig :latent$_A E@cref$

:fig :latent$_V AE@cref$

:fig :latent$_A E@cref$

:fig :latent$_V AE@cref$

:fig :latent$_A E@cref$

:fig :latent$_A E@cref$

?? , the goal is to obtain a latent space organised such as in **??**.

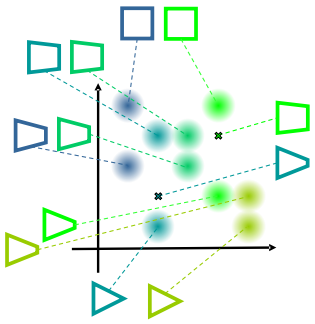

FIGURE 2.20 – A representation of a disentangled latent space with two generative factors, shape and colour. The space is even more organised that for VAE. Moving along one dimension should only change one of the characteristics.

2.3.1 Non-Deep Learning Approaches

The first attempt in disentanglement in time series predates the apparition of deep learning and latent space. The challenge of source separation, a classic of signal processing, can be seen as a disentanglement problem. From a recording that corresponds to a

mix of several signals (for example two people talking at the same time), the goal is to separate the original signal (each signal is a clear generative process, coming from different sources).

Independant Component Analysis (ICA) is one of the oldest representation learning algorithm [**ICA1984** ; **JUTTEN1991**]. It came from a background of dimensionality reduction. It aims at separating a signal into several independent underlying signals that sum to the observed signal. Because ICA uses linear modelling, it can only separate linearly into underlying subsignals. There have been many improvements and variations on the original algorithm [**reviewBSS**]. The most notable is Independant Vector Analysis (IVA), which extends the framework to allow for multivariate (and therefore vectorized) signal [**IVA**].

In 1999, another method called Non-Negative Matrix Factorization (NMF) has appeared, presented by **NMF**. This method transforms the input signal in a representation with two axes : time and frequency (like a spectrogram). It then decomposes this signal in two parts : a dictionary of patterns present in the signal in frequency (each pattern is called an *atom*) and an activation matrix that characterise these patterns in time.

This decomposition transforms each example in the dataset in a weighted sum of the learned atoms. Because the matrices (dictionary and activation) are non-negative, the weighted sum has a sense and experts can analyse the different atoms and explain their signification. The main advantage of this approach over VAE and GAN is the ability to add constraints that can be enforced on the space of the atoms (they can be forced to be orthogonal for example).

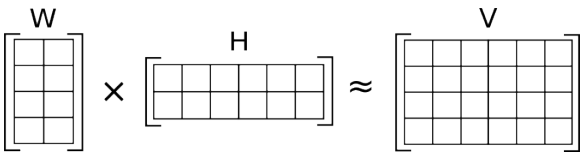

FIGURE 2.21 – Illustration of non-negative matrix factorisation : the matrix V is represented by the two smaller matrices W (the matrix of weight, which can be seen as the activation) and H (the matrix of atoms, which can be seen as a dictionary), which, when multiplied, approximately reconstruct V. Credits to wikipedia **wiki**

NMF was also improved on in various ways [**reviewBSS**] ; many objective functions were explored and several training methods can be used. One notable improvement is Multichanel Non-Negative Matrix Factorization (MNMF) [**MNMF**], which, like IVA for ICA, allows NMF to be used with multivariate signals.

2.3.2 Deep Learning Approaches

As explained before, deep generative models have great potential for learning structured latent spaces. They have been modified to perform constrained generation, which is a generation task where the output is conditioned. If this condition is made on real world

concepts, this is akin to disentanglement. Both types of architecture of generative models (adversarial and VAE) have been adapted for this purpose. Other deep learning methods, not necessarily related to generative models, have been used to perform disentanglement. Most of these techniques have been used in combination with either VAE or GAN (mostly the latter, because of its high flexibility). **??** we will first present the variations and adaptations of the adversarial techniques to perform disentanglement. **??** will then present the variations of VAE on the same goal. **??** will present another deep technique used to perform disentanglement called Representation Mixing. This technique is usually used in combination with adversarial techniques. It is singled out from the other techniques because it is the one we use in our work. Most of the work on disentanglement using deep learning has been conducted on images. We worked on time series, which are quite different. **??** will present deep learning techniques that have been used for disentanglement on video and speech. These two media are closer to our work. Indeed, although videos are made of images, they have a sequential component. Speech is, in essence, a time series and therefore much closer to our work.

2.3.2.1 Variation on GANs and adversarial techniques

Adversarial autoencoders The first example of these models is called Adversarial Autoencoders [**makhzani2015adversarial**]. It mixes the adversarial approach of GAN with a classic AE. Much as in VAE, the goal is to impose a structure on the latent space. The idea is to use an adversarial discriminator to impose a specific distribution over the encoded vectors. The discriminator $discr$ learns to differentiate between an encoded representation $e(\mathbf{x})$ and the chosen distribution. The decoder d optimises the reconstruction loss between an example $\mathbf{x}$ and its reconstructed version $d(e(\mathbf{x}))$. The encoder e must balance these two objectives by encoding the most information about the input while respecting the prior. **??** shows a representation of the model.

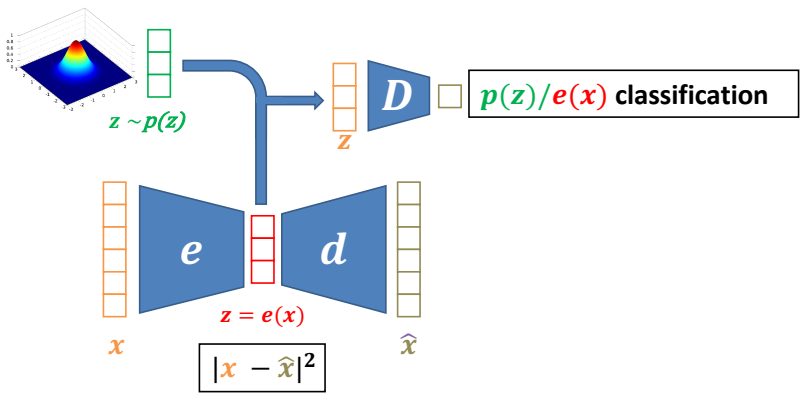

FIGURE 2.22 – A representation of an adversarial autoencoder with a L_2 loss. The addition of a simple adversarial constraint on the distribution of the latent vector allows for some disentanglement.

It is expressed with the following objective functions :

$$\max_D \mathbb{E}_{z\sim p_z}[\log\ D(\boldsymbol{z})] + \mathbb{E}_{\mathbf{x}\sim p_\mathbf{x}}[\log\ (1 - D(enc(\mathbf{x})))] \tag{2.13}$$

$$\min_{e,d} \mathbb{E}_{x\sim p_\mathbf{x}}[||x - d(e(\mathbf{x}))|| - \log\ D(e(\mathbf{x}))] \tag{2.14}$$

This method enforces a structure in the latent space similarly as what is done in VAE. In **??**, you can see a comparison of the latent space of a VAE and the one of an adversarial AE when using a Gaussian prior. Both latent spaces are clearly well structured but the Adversarial AE seems to have a better one.

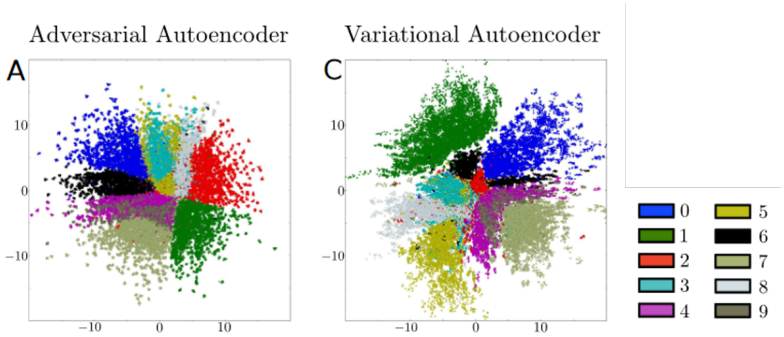

FIGURE 2.23 – A comparison of the latent spaces of an Adversarial Autoencoder and a VAE when the prior is a 2D Gaussian. Credits to **makhzani2015adversarial**

Moreover, the chosen distribution can be used to perform a kind of unsupervised clustering. For example, for one of the well-known dataset MNIST (of handwritten digits), the constraint could be that the encoded vectors must follow a mixture of ten Gaussians (one by class of digits). This would be akin to perform clustering in the latent space of representation and hope it would encode each digit class into one of the Gaussians. Though this prior can be used in classical VAE, it does not result into the same latent space, as shown in **??**.

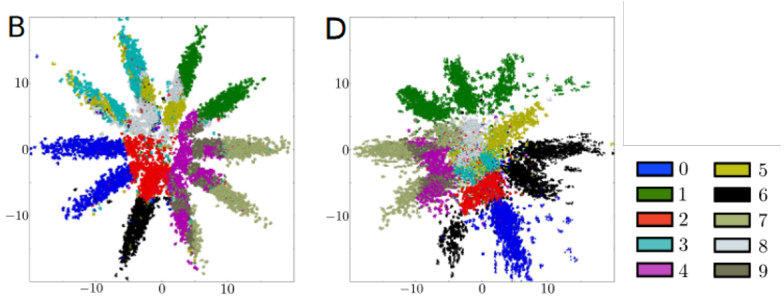

FIGURE 2.24 – A comparison of the latent spaces of an Adversarial Autoencoder and a VAE when the prior is a mixture of 2D Gaussian. Unfortunately, the 10 Gaussians do not match the 10 classes. Credits to **makhzani2015adversarial**

The hope that each class of numbers would be encoded into one of the Gaussians of the mixture is not fulfilled and disentanglement is not achieved. Some supervision has

to be added (the use of a fraction of the labels was necessary so that the cluster would match the class of digits) so that true disentanglement appeared. The results can be seen in **??**.

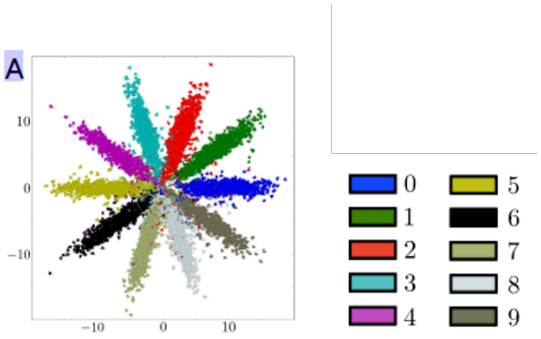

FIGURE 2.25 – The latent space of an adversarial autoencoder with a prior of a mixture of 10 2D Gaussians and the used of a subset (around 20%) of the label to enforce one Gaussian by class. Credits to **makhzani2015adversarial**

Censoring Using AE and Adversarial Techniques Approaches where the goal is to remove some information from a representation are called censoring. They were first described by **EdwardsS15**. Removing information from a representation can be very useful when considering the fairness of a model because it is then able to truly ignore some attributes of its input.

One such model based on AEs and adversarial techniques is the *fader network*. Presented by **lample2017fader**, the idea is to learn a latent space where the attributes we want to control and conditioned on have been destroyed. Instead, these are given directly to the decoder that must only take this input into consideration. To remove the information about specific attributes (such as gender), a discriminator is learned on the latent space that tries to extract the attribute. Using techniques such as gradient ascent for the discriminator forces the encoder to output a vector that does not contain the attribute the discriminator is trained to recover. This forces the latent space to hold no information pertaining to this attribute. As an example, when autoencoding faces, most models will naturally encode gender. To remove this attribute from the representation, fader networks required the discriminator not to be able to accurately classify the gender of an example only from its representation. The encoded representation is then given to the decoder along with a binary information that indicated which gender to reconstruct.

Conditional GANs This improvement of GANs was made in several steps. **mirza2014conditional** made the first step, when they introduced a code variable $\boldsymbol{c}$ so that their Conditional Generative Adversarial Network (CGAN) would model the joint distribution $p(\mathbf{x}, \boldsymbol{c})$. Using the identity : $p(\mathbf{x}, \boldsymbol{c}) = p(\mathbf{x}|\boldsymbol{c})p(\boldsymbol{c})$, they introduced the conditional generator G that adds to the classical input $\boldsymbol{z}$ the code $\boldsymbol{c}$. This generator can be thought as modelling the conditional probability $p(\mathbf{x}|\boldsymbol{c})$. The discriminator also adds to its input the code $\boldsymbol{c}$ and must differentiate between pairs of$(\mathbf{x}, \boldsymbol{c})$ and pairs of $(G(\boldsymbol{z}, \boldsymbol{c}), \boldsymbol{c})$ where $\mathbf{x}$ come from the data-

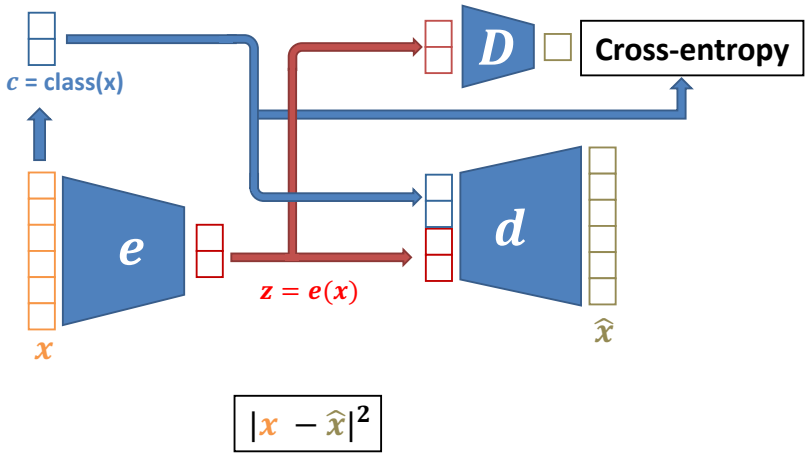

FIGURE 2.26 – The fader networks model uses AE and adversarial techniques to perform representation censoring.

set and z is sampled according to a prior. An illustration of the model is shown in **??**. The generator G takes as input the code $\boldsymbol{c}$ and the prior z and outputs a generated sample. The discriminator D takes as input the code $\boldsymbol{c}$ and an example $\mathbf{x}$ (either generated by G or from the dataset) and classifies the pair as either real or fake.

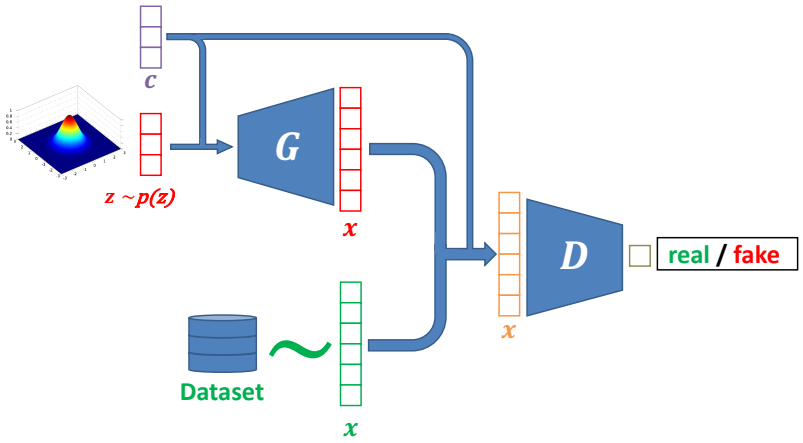

FIGURE 2.27 – The Conditional GAN model adds a code $\boldsymbol{c}$ that controls an attribute of the generated sample

This leads to the minimax formulation :

$$\min_G \max_D \left[\mathbb{E}_{\mathbf{x},\boldsymbol{c}\sim p_{\mathbf{x},\boldsymbol{c}}}[\log\ D(\mathbf{x},\boldsymbol{c})] + \mathbb{E}_{z\sim p_{\boldsymbol{z}},\boldsymbol{c}\sim p_{\boldsymbol{c}}}[\log\ (1 - D(G(\boldsymbol{z},\boldsymbol{c}),\boldsymbol{c}))] \right] \tag{2.15}$$

This first model proved to be very hard to train. This is why **odena17a** introduces the Auxiliary Classifier Generative Adversarial Network (ACGAN). They added an auxiliary objective of classification for the discriminator. The model is illustrated on **??**. This method greatly improves stability. However it also biases the model toward samples that are easy to classify. Some research was conducted on how to alleviate this problem, with

miyato2018cgans removing the auxiliary classifier and instead constraining the architecture, and **NIPS2019_8414** adding yet another classifier to minimise mutual information between $\mathbf{x}$ and $\boldsymbol{c}$.

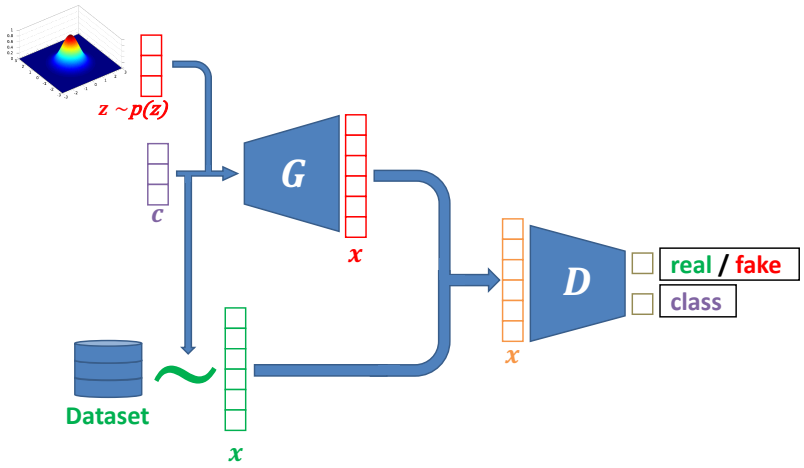

FIGURE 2.28 – The ACGAN model adds a classifier to the classical GAN in order to control the class of the generated sample.

InfoGAN This information mutualisation objective was used in another approach. Information Maximizing GANs were presented by **infogan**. They added a regularisation based on mutual information to the GAN objective function. They split the input of the generator in two parts, one traditional compressible noise $\boldsymbol{z}$ and one *latent code* $\boldsymbol{c}$. The goal is to make the content of $\boldsymbol{c}$ meaningful, so that it matches a specific characteristic of the sample. They achieve this by maximising the mutual information between the code and the generated sample. Because computing the mutual information is often impossible, they use a classical variational method by adding an auxiliary distribution $Q(\boldsymbol{c}|\mathbf{x})$ that must approximate the real distribution of $p(\boldsymbol{c}|\mathbf{x})$. The InfoGAN model is illustrated in **??**.

Because the mutual information is not computable, they used a lower bound :

$$L_1(G,Q) = \mathbb{E}_{\boldsymbol{c}\sim p_c, x\sim G(\boldsymbol{z},\boldsymbol{c})}[\log\ Q(\boldsymbol{c}|\mathbf{x})] + H(\boldsymbol{c}) \tag{2.16}$$

where $H(\boldsymbol{c})$ is the entropy of the latent code. Because its distribution is fixed, it is considered a constant.

This gives the following objective function for InfoGAN :

$$\min_{G,Q}\max_{D} V_{InfoGAN}(D,G,Q) = V(D,G) - \lambda L_1(G,Q) \tag{2.17}$$

where $V(D,G)$ is the classical objective of GAN.

In practice, it means that a third neural network is added to the model. This one often takes as input the last hidden layer of the discriminator (so a representation of the

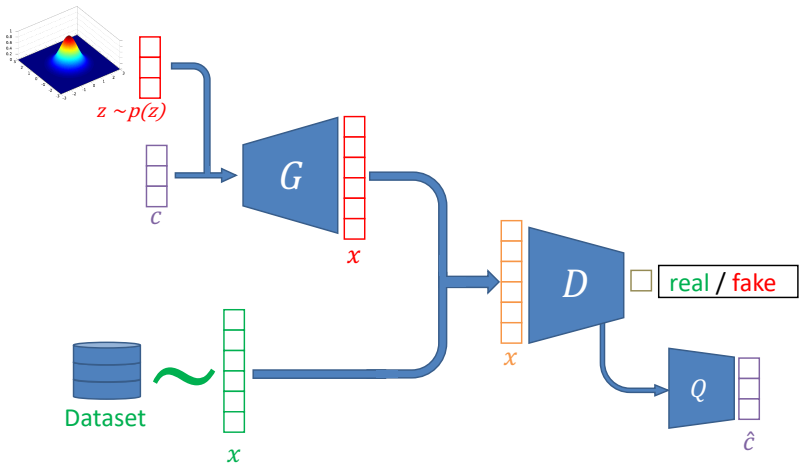

FIGURE 2.29 – The infoGAN model that uses information maximisation between the generated sample and the code used as conditional input without supervision

generated sample skewed toward deciding if it was generated or not) and outputs the likelihood of c given the generated sample.

This means that the statistics of the probability distribution must be fixed. In the original article [**infogan**], for the experiments on MNIST, they were three latent codes. One latent code represents the class. Its chosen distribution is uniform on a discrete set of 10 values (hopefully the code would match the classes). In this case, Q is simply one layer of fully connected with a softmax non-linearity. The two other codes are both continuous on $[-1, 1]$. One represents the rotation of the number, the other the width of the line. Both chosen distributions were Gaussian. The outputs of Q were the mean and standard deviation and the re-parametrisation trick is used to be able to back propagate the gradient.

2.3.2.2 VAE variation

Just like GANs, VAEs were modified and improved to allow for conditional generation.

Conditional VAE The first of this variation was Conditional Variational Autoencoder. Introduced by **conditionalVAE**, they modified the classic VAE with the addition of a code $\boldsymbol{c}$, given to both the encoder and the decoder. The code $\boldsymbol{c}$ can be either a one-hot vector to indicate a class to generate or a part of an image which has to be completed. The goal is for the encoder to now model $q_\phi(\boldsymbol{z}|\mathbf{x}, \boldsymbol{c})$ and the decoder to model $p_\theta(\mathbf{x}|\boldsymbol{z}, \boldsymbol{c})$. This is done by slightly modifying the lower-bound so that it is on the log-likelihood of $p(\mathbf{x}|\boldsymbol{c})$.

$$L_{CVAE} = \mathbb{E}_{q_\phi(\boldsymbol{z}|\mathbf{x},\boldsymbol{c})}[\log\ p_\theta(\mathbf{x}|\boldsymbol{z}, \boldsymbol{c})] - KL(q_\phi(\boldsymbol{z}|\mathbf{x}, \boldsymbol{c})||p_\theta(\boldsymbol{z}|\boldsymbol{c}))$$

With this formulation, the prior $\boldsymbol{z}$ should now be conditioned on $\boldsymbol{c}$. However, the authors simply decided to model $\boldsymbol{z}$ and $\boldsymbol{c}$ to be independent such that $p_\theta(\boldsymbol{z}|\boldsymbol{c}) = p(\boldsymbol{z})$.

This choice works well when the code is a category (one-hot vector) but seems to be an obstacle when it is an image. It is probably because it becomes impossible for the model to learn a prior on z that is independent of c, which means that the decoder cannot process the independently sampled z and c it is given as input.

β-VAE Probably the most interesting improvement of VAE in terms of disentanglement, β-VAEs were presented by **Higgins2017betaVAELB**. They noticed that using a diagonal Gaussian prior helps the model to discover which dimensions are independent of each other, which is close to performing disentanglement. Intuitively, this is due to the fact that the KL pushes the posterior to learn dimensions independent of each other because the prior has all its dimensions independent of each other. Their idea was to strengthen this tendency by adding a weight to the KL term of the objective function. By simply adding an hyperparameter β that controls the weight of the regularisation in the objective function, they achieved results as good as infoGANs'. The objective function is thus :

$$L_{\beta-VAE} = \mathbb{E}_{z\sim q_\phi(z|\mathbf{x})}[\log p_\theta(\mathbf{x}|z)] - \beta \cdot KL(q_\phi(z|\mathbf{x})||p(z)) \tag{2.18}$$

At the time, not all phenomena involved in this 'natural' disentangling behaviour of VAEs was understood. Since then many theoretical work was done to understand and explain it better [**dai2018diagnosing**].

FactorVAE and β-TCVAE Building on these results, two concurrent works [**abs-1802-04942** ; **kim2018disentangling**] transformed the objective function to make the Total Correlation term appear. The goal is to penalise this term as it measures the dependencies between each dimension of the posterior. They achieved very good performances and established state-of-the-art results for unsupervised disentanglement.

2.3.2.3 Representation Mixing

This technique of disentanglement is the one we used in our work. The idea first appeared in **MathieuZSRL16** with the use of a mix of adversarial techniques and VAE. The goal is to learn to encode each attribute we want to factorise and then mix these representations in order to decode them into a new example. For example, if we denote (z_1, s_1) the representations of two attributes of $\mathbf{x}_1$ and (z_2, s_2) the representation of the same attributes for $\mathbf{x}_2$, then the generation of two new examples via the decoding of (z_1, s_2) and (z_2, s_1) is quite simple. In **MathieuZSRL16**, each example $\mathbf{x}$ is only labelled with one class y. The factor s corresponds to this class while z corresponds to some other attributes (unspecified). They also added two adversarial discriminators in order to ensure that the representation s encoded the class (and not something else). A representation of the model can be found in **??**.

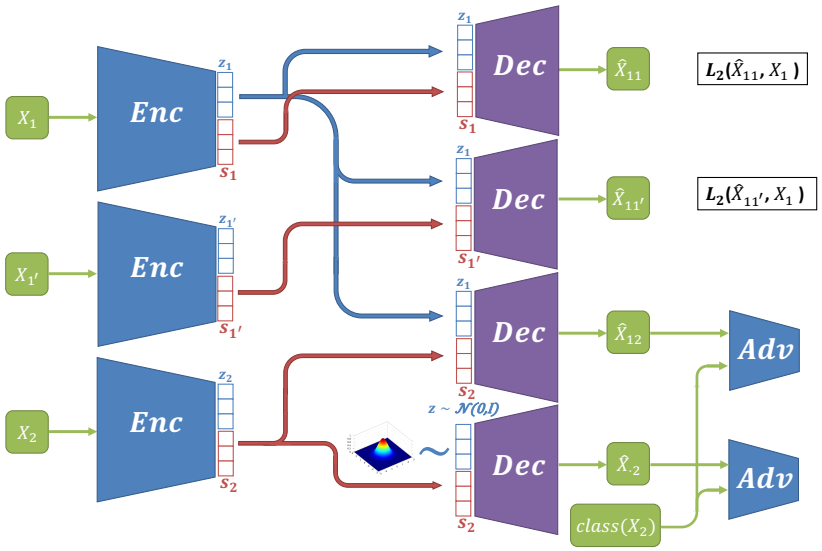

FIGURE 2.30 – The model of **MathieuZSRL16**. The encoder is actually probabilistic for z, outputting a mean and variance that are used to create a Gaussian that is sampled to obtain the z of the decoder. The upper decoders are used to enforce the reconstruction loss, while the bottom ones are used in combination with the adversarial part to ensure that class is indeed encoded into s. As usual in adversarial techniques, the training alternate between encoder/decoder and discriminator.

2.3.3 Conclusion

Because most of the research on disentanglement has been done on computer vision tasks, it is difficult to compare our approach directly to the existing state of the art. The work on disentanglement in temporal settings are mostly in the research on spatiotemporal time series created by physical processes and in generation and analysis of speech and videos. We will present the most recent work on these subjects to illustrate the abundance of recent work on using disentanglement with time series.

Physical Processes The study of physical processes has been a focus of recent research. Because we know they obey clearly defined rules (such as the conservation of energy), the formulation of these problems often leads to the resolution of differential equations. It is especially present in meteorological challenges where the use of Neural Ordinary Differential Equation (ODE) has become popular. They are often used to forecast spatio-temporal physical problems [**Pajot**; **yldz20190de2vae**; **Chen2018NeuralOD**]. However, it is only recently that the idea of learning disentangled representation of the spatial and temporal contexts has been implemented with a Partial Differential Equation (PDE) resolution method [**don2020pdedriven**]. This work resembles our own as it uses a representation mixing and AEs but the difference lies in the differential hypothesis made on the input series.

Videos and Speech Videos are somewhere between computer vision and time series and therefore very good candidate for the transfer of disentanglement techniques from computer vision tasks to time series modelling. The techniques used with videos are usually based on VAE approaches and adversarial techniques.

The most common approaches for conditional generation of video are quite similar to the techniques presented in the computer vision task. Most of the time, the idea is that one must disentangle the visual aspect of a video from its dynamic aspects. It is often based on autoregressive models where RNN are trained at the image level. Using adversarial training, **Villegas2017DecomposingMA** and **DentonB17** focused on predicting the next frame of a video by using different AEs architectures and disentangling dynamics and content. On one hand, **Villegas2017DecomposingMA** learned to represent the motion by comparing two consecutive frames. On the other hand, **DentonB17** designed a new training objective to learn disentangled representations of content and dynamics on only one frame. Other works, such as **franceschi2020stochastic** also models the dynamic of the video in a state-space manner while the content is encoded outside of the RNN.

The closest domain we could find where disentanglement was used in research is speech. Speech is a univariate time series and it has been the focus of many studies; either with classification for speaker recognition or for disentanglement with source separation. As an example, **yingzhen18a** designed a model based on VAE specifically for sequential data. Similarly as the approaches described above, they disentangled between features that are time dependant (dynamic) and features that are preserved over time (content). They are able to condition generation on one or the other. They tested their model on both video and speech. For speech disentanglement, the content is the sentence being pronounced and the dynamics the voice saying it. This allows them to perform 'feature swapping' and generate speech with a voice different from the one of the original speaker. In the context of speech recognition, **hsu2017unsupervised** presented the Factorised Hierarchical Variational Autoencoder (FHVAE). This model is trained by adding a discriminator to the ELBO that regularise the VAE to enhance its disentangling capabilities. They explicitly learn two priors, one independent of the sequence, the other one dependant by making use of the multi-scale nature of speech data, where one can work at sequence level and at segment level.

Time Series and Disentanglement Time series and disentanglement are two domains that have been studied intensively. However, they have rarely been studied together. Studies of disentanglement have mostly focused on computer vision tasks and were in their vast majority conducted in the last decade. Though impressive results were obtained, they are not easy to transfer to time series. Time series, on the other hand, have been studied for over a century. Many tasks have been researched : forecasting, classification, imputation... Forecasting and classification in particular have been extensively researched because of their wide range of applications. Though some work has been done on combining the two domains, no silver bullet has been found. Just like in computer vision, the techniques used range from variational inference to adversarial techniques, passing by representation mixing and AE. Each technique has its advantages and drawbacks and,

depending on the form of the input, one can be more suited than the other. VAEs have a strong power of disentanglement but generate outputs that often dismiss high-frequency details. On the other hand, adversarial techniques such as GANs can generate realistic samples and are very flexible but they often fail to capture different modes. They often devolve into modelling only the examples that fall into main categories and not the rest which are not necessarily outliers. Representation mixing can incorporate either of these approaches to perform disentangled representation learning. However, it also requires more supervision than the other approaches. As we knew we could obtain a somewhat strong supervision on contexts, we decided to use representation mixing.

2.4 Conclusion

In this chapter, we presented an overview of the different domains our work drew inspiration from. Starting with the ancient method of time series modelling, we showed how the evolution of the domain led to a more data-centric approach, and less use of domain experts' knowledge. We presented the different existing models for learning meaningful, disentangled latent spaces that can be classified in different categories, spanning from dimensionality reduction approaches (such as AEs) to conditional generation (such as InfoGAN), passing by the strictly disentangling approaches (such as source separation).

CHAPITRE 3

TIME SERIES'S GENERATION AND FORECASTING FROM DISENTANGLED REPRESENTATIONS OF CONTEXTS

Contents

Introduction

The rise of Machine Learning (ML) and specifically Deep Learning (DL) has led to major data collection efforts. Typically, these techniques require (very) large amount of data to be collected and most of the time annotated in order to be effective. This has led to the creation of many large datasets in the last decade. However, in many cases, collecting real world data (labelled or not) is very expensive. Often, available data will focus on specific aspects of the quantity that is collected. For instance, we could have very precise measurements for only some conditions (a set of a few discrete locations and days for example). We believe that with rich enough representation of context, the tasks of generation and forecasting will become very similar. Indeed, with conditional generation, if the condition contains enough information, the model should not need any other data to predict the entire time series.

The model we present in this chapter aims at leveraging the limited available data (via either forecasting or generation) and extrapolate on conditions where data was not collected. By learning rich enough representations of contexts, our model is able to predict time series for combinations of contexts where they haven't been measured.

We made the hypothesis that, in some cases, users can be considered like a context. With this in mind, we can draw a parallel with recommender systems. In collaborative filtering, when we have representations of users and products, a simple scalar product leads to good recommendations. One of the known issues of this method is the cold-start problem. To add a new product or user, most of the time, the systems need to be retrained almost from scratch. We aimed at learning good representations of contexts, with the secondary goal of being able to add new context after training. This is akin to

few-shot (and even one-shot) learning where we aim at learning or adapting a model from very little data [**palatucci2009zero**].

This is why our model can tackle the extreme case where data corresponding to new conditions is collected after training. From a unique time series corresponding to a new context, our model can learn a representation that can then be used in combination with the other learned representations.

We achieve this result by learning disentangled representations of the different contexts that gave rise to a specific time series. We use a method called representation mixing with a specific training procedure to ensure disentanglement. We tested our model in two settings. The first one is centred around missing data in the case where one example is an entire time series. The second one focuses on the cold-start problem and the introduction of new contexts after training. We used four datasets in our experiments. We chose them because they have clearly labelled contexts and, as they were created by the aggregation of daily human activity, they are very sensitive to context. They could all be considered as spatiotemporal time series as their labelled contexts are location and time.

Our model was designed incrementally and has three versions. The first version cannot tackle the cold-start problem. The second, version is able to perform this task but adds an aggregation problem. Our third version uses an attention mechanism that solves this aggregation problem and adds some stability to the model. In this chapter, the section 3.1 presents a review of related works for spatiotemporal time series and attention mechanisms. The section 3.2 presents the tasks we defined and the three versions of our model we used to tackle them. Finally, the section 3.3 presents the experiments we made to test our models and their results.

The work in this chapter has led to the publication of two conference papers :

- **esann (Best student paper award)**

- **itsc**

3.1 Related work

As we showed in chapitre 2, time series prediction and understanding are historical issues in machine learning [**box1968some**]. Despite steady progress over the years, many challenges remain, ranging from finance and market prediction to user modelling and recommender systems, from smart cities and transportation systems to supply chain and sales prediction.

The datasets we focused on in this chapter all have a spatiotemporal component. In fact, space (location) and time (day of the year) are the two contexts we study. We wish to learn representations of these contexts so that entire time series can be inferred from them.

This is why sous-section 3.1.1 will present an overview of related works on spatiotemporal time series. sous-section 3.1.2 will present an overview of attention mechanisms in the literature, to place our use of the mechanism in context.

3.1.1 Spatio-Temporal Series

The literature contains a large amount of research dedicated to spatiotemporal time series. Several models have been used to tackle them.

Whereas most approaches are fully dedicated to the time series itself, it appears that contextual information has a major influence on the signal shape. Public transportation flows mainly depend on stations and weekdays ; pollution levels in a city are directly related to the location of the measurement and the weather conditions ; sales – and the necessary inventories – are strongly influenced by marketing campaigns and seasonality as was shown by **samal2019time**.

Historically, taking into account the context was the responsibility of a human analyst in most applicative domains. Several attempts to overcome this limit have been proposed in the last decade, such as the work of **si2013exploiting** in the field of finance or of **farajidavar2017deep** in the field of public transportation. Deep learning clearly represents a major breakthrough in the field. **LSTM** developed recurrent architectures that offer great opportunities to model the dynamics while allowing for the inclusion of graph constraints as was shown by **Ali** ; **bourigault2014learning**. Spatio-temporal time series can sometimes be seen as multivariate time series. In this case, each time step is represented the aggregation of the values for each location. **karim2019multivariate** created a model specifically to treat such multivariate time series. However, we believe that in this case, most information from the contexts is lost. Using datasets focused on weather, **Liu2016** developed spatiotemporal neural network architectures. They encoded spatial relationships simply via Euclidean distance. This method allows them to connect the different locations and use this information in their model. **Ali** proposed to encode the relationships between locations in a matrix of 0 and 1 or even to learn that matrix, allowing for more complex graphs to connect the contexts.

In weather, however, most of the time, the locations are placed according to a very regular grid. In our work, the location context does not follow a rigid pattern.

One of the datasets we use is focused on transportation data, and some works have already been done using it (or other similar datasets). We therefore present a rapid overview of works specifically related to public transportation.

Since the appearance of massive public transportation data, researchers have exploited it to resolve different issues specific to transportation systems. A review is available in **Pelletier2011**. Smart cards automated fare collection systems (AFC) have replaced ground surveys in order to understand users' behaviours and track their habits. **Zhao2017** computed users' trip patterns thanks to origin and destination logs and are able to detect interpretable behaviours and to cluster users. **Tonnelier2016** provides latent users' features via matrix factorisation and achieves similar goals.

Anomaly detection, which is also a common issue in time series and spatiotemporal time series, is tackled by [**Tonnelier2018**; **Izakian2014**] who presented two different methods. **Izakian2014** are able to visualise the propagation of anomalous behaviours, i.e. different from the normal periodic trend. Spatially adjacent time series are separated in consecutive time windows which are attributed an anomaly score computed via the deviation from the previous window or from a reference value.

3.1.2 Attention Mechanism

An attention mechanism is a part of the model that assigns weights to an input in order for the network to focus only on parts of the input. This allows focusing on the salient features of the input. The salient features are quite often a matter of context and can change, depending on the input, the output and the context. This is why attention mechanisms often change with time and/or context.

Before its current formal definition (the one we will use), several mechanisms with the same goals already existed. For example, **Graves13** generated handwritten text from text data and used a window of weights so that the network would focus on letters one by one, drawing one before going to the next one. However, in this case, the window would only move forward as the handwritten text is generated.

There are cases where some latter inputs have to be focused on early on and some earlier inputs have to be read later on. For example, in machine translation, the order of the words in one language can be the opposite of the order of the other language.

This was the motivation behind the work of **bahdanau2014neural** in which they presented a formal definition of an attention mechanism. They introduced a mechanism in which each hidden state is assigned an alignment score that becomes the attention weights through the use of a softmax. The multiplication of the attention weights with the hidden state allows the model to learn which part of the input it should pay more attention to.

Since then, this mechanism has been declined and improved in many ways, the most notable one being presented by **LuongPM15**. By slightly modifying the manner in which the alignment score is calculated and by changing the position at which the attention mechanism is, they make it both easier to train and less computationally expensive.

Other variations of the mechanism exist, the goal being for the network to focus on parts of the input and not take everything as is. An attention mechanism also helps with regularisation. As only part of the input is used after the attention mechanism, it is akin to feature selection. However, contrary to what is done with the usual methods of feature selection, the features selected will depend on the input. This mechanism keeps the regularisation power of feature selection while remaining extremely light. Very few parameters are needed to implement such a mechanism.

The specificity of such mechanism also lies in the fact that it has to be used during inference. Unlike other regularisation methods, it is essential to the model because it is sequential by nature. A first representation is learned, then the attention mechanism focuses

on a specific part of this representation and then a decision is taken. This sequentiality is reminiscent of Reinforcement Learning (RL) methods.

These mechanisms usually improve the modelling of dynamics and contextual factors. This in turn leads to a better understanding of the object to process. Because the weights associated with the attention mechanism can be accessed, this sometimes makes the model more explainable. Being able to see what the model is focusing on at any point during inference or training helps to understand its process.

In 2017, **vaswani2017attention** presented the Transfomers : a new model using Multi-Headed Attention designed for machine translation. Though they also use a attention mechanism, it is quite different from the one presented above. As this work only use the simpler, older version, we will not go into details on this subject.

3.2 Model

This section presents our encoder-decoder based models as well as the tasks it was designed to solve. Our three models aim at learning rich representations of contexts. They learn disentangled representations of these different contexts in order to generate or forecast time series. From these contexts' representations, they are able to predict behaviour for an unseen combination of these contexts. Two of our models can also perform a specific type of one-shot learning, by representing a new factor while only seeing it once during inference (after training). Each model is slightly different, which allow them to tackle different tasks and settings. For clarity, all examples are given for datasets in which there are two distinct factors to disentangle, although our model could accommodate more.

3.2.1 Notation and Tasks

We consider the problem where one has access to a set of time series $\mathbf{x}_{i,j}$ labelled with values of factors of variations characterising the observed series. Here, there are two factors of variation i and j [1].

When considering for instance a city subway network, one example $\mathbf{x}_{i,j}$ could be the number of ticket validations each hour for one day i in a particular subway station j.

Such a dataset (organised along two factors) can be represented as a matrix in which each cell corresponds to a time series. The columns of the matrix represent one factor (factor 1), the rows the other one (factor 2). In our case, each time series is univariate and of fixed length T such that $\mathbf{x}_{i,j} \in \mathbb{R}^T$. However, the model, in its recurrent version, can easily be extended for series of different lengths, and any version can be adapted for multivariate time series.

1. Though our model could be adapted to handle more than two factors of variation, for the sake of clarity, we will limit ourselves to two factors of variation.

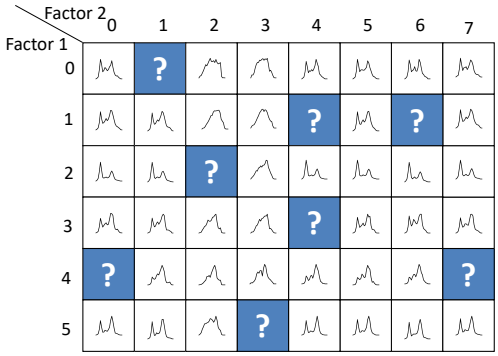

FIGURE 3.1 – Our first task : missing series prediction. Every possible class of each factor has been observed in the training set. The time series on white background represent the training set. The question marks represent the test (and validation) set ($m_{i,j} = 0$).

We call each possible value of a factor of variation a factor class, and the integer associated to it is called factor class value. We consider that training series are provided for specific combinations of $i \in [\![1 \,..\, N]\!]$ and $j \in [\![1 \,..\, M]\!]$ as illustrated in :fig :task1@cref

:fig :task1@cref

:fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task1@cref

:fig :task2@cref

:fig :task2@cref

:fig :task1@cref

3,2,2 :fig :task2@cref

3,2,2 :fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task1@cref

?? .

The observed series are indexed through a mask m such that $m_{i,j} = 1$ if $x_{i,j}$ is observed (and is in the training set), and $m_{i,j} = 0$ if $x_{i,j}$ is not observed.

The two tasks we identified are

MISSING SERIES PREDICTION The objective of this task is to predict the values of the temporal series that are not in the training set i.e. the values of all the $\mathbf{x}_{i,j}$ with

$i \in [\![1 \,..\, N]\!]$ and $j \in [\![1 \,..\, M]\!]$ such that $m_{i,j} = 0$ but there exist at least one k, k' such that $m_{i,k} = 1$ and $m_{k',j} = 1$

This task can be seen as a matrix completion problem where missing entries are temporal series. It typically applies to applications, where, for cost reasons, time series have not been collected for all combinations of contexts.

This task is illustrated in Figure 3.1.

NEW FACTOR SERIES PREDICTION The objective of this task is to predict series for new factor classes i.e. new values of i or j that do not appear in the observed set of series. It corresponds to a setting where one observes a new series [2] $\mathbf{x}_{i,j}$ such that $j \in [\![1 \,..\, M]\!]$ but where $i \notin [\![1 \,..\, N]\!]$. For instance, a series for a new subway station that has never been observed in the past. In this case, the goal is to predict all the missing series for that particular factor's value i, i.e. all the $\mathbf{x}_{i,k}$ for $k \neq j$.

This task is illustrated in Figure 3.2.

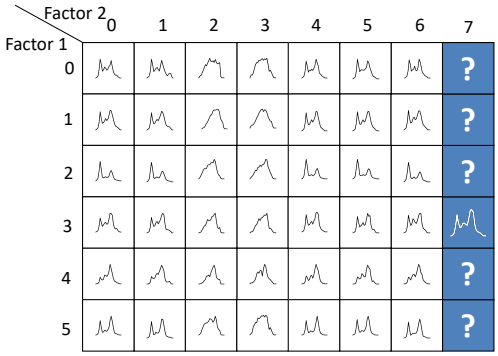

FIGURE 3.2 – Our second task : new factor series prediction with a new factor 1 class seen at inference. The time series in white background represent the training set. The question marks represent the test (and validation) set. On key difference from the other task is that there is a series that is not in white background but is also not a question mark. This represents the series that is seen for the first time during inference and will allow us to predict the series in the test set.

Both settings can be solved if we are able to capture from the training set the influence of each factor i and j in $\mathbf{x}_{i,j}$.

Our model was incrementally improved to tackle different settings and tasks. The first version, we called transductive, can only perform our first task : missing series prediction. The second version, we called inductive, can tackle new factor classes but adds an aggregation challenge that our third version (called with attention) resolves while adding the possibility to work on noisy data.

The principle of our approach is to compute a representation of each factor of variation in a latent space, and to use these representations to infer any series values. Let us consider two latent spaces $\mathbb{R}^{\ell_1}$ and $\mathbb{R}^{\ell_2}$ of dimensions ℓ_1 and ℓ_2. Each value of the 1^{st}

2. It also applies to the problem where $x_{i,j}$ is the newly observed series.

factor i (respectively of the 2^{nd} factor j) will be encoded into a vector $\boldsymbol{z}_i^{(1)}$ (respectively $\boldsymbol{z}_j^{(2)}$). Based on these representations, the series $\hat{\mathbf{x}}_{i,j}$ will be inferred through a decoder function. The sizes of the latent spaces ℓ_1 and ℓ_2 can be different. In many cases, the factors represent very different contexts and one context may contain more information than the other one, and therefore requires a bigger representation. For example, one of our datasets has 90 possible values for a factor and 300 for the other. The latent spaces should therefore be of different sizes. Moreover, a smaller representation for a factor containing less information can serve as a regularisation mechanism.

The encoder is used to get the disentangled representation of the factors ($\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$). It is actually comprised of several encoders (one for each factor) that may share some parameters. The decoder takes the representations of the factors and outputs a reconstructed time series. The decoder is the same for each version of our model and it can be adapted to perform either forecasting or generation.

The next section describes for each version, its parts, its training procedure and its inference usage.

3.2.2 Transductive model

This version of our model is described in :fig :model$_emb@cref$

:fig :model$_emb@cref$

:fig :model$_emb@cref$

:fig :model$_emb@cref$

?? . It can only tackle our first task : missing series prediction. The model is comprised of two parts, an encoder and a decoder. Our encoders are actually embedding matrices and not learned parametrised functions.

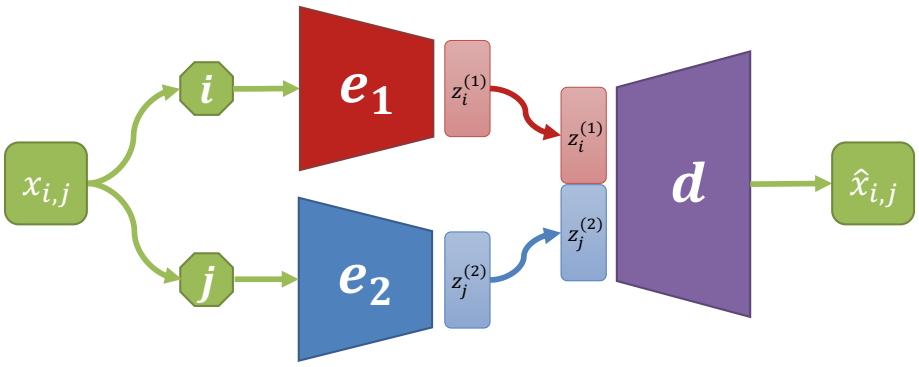

FIGURE 3.3 – The transductive setting, where the encoders are learned embeddings, taking integers as input, leading to a unique representation by factor class.

3.2.2.1 Encoder

In this model, our encoder takes as input the factors' class values (integers) and outputs one representation for each factor. They are in fact embedding matrices and not parametrised function. Let us consider $Z^{(1)}$:

$$Z^{(1)} : [\![1 \,..\, N]\!] \to \mathbb{R}^{\ell_1} \tag{3.1}$$

$$u \mapsto \boldsymbol{z}_u^{(1)} \tag{3.2}$$

a function such that given an integer $\in [1, N]$ is able to compute the representation of the first factor of this factor class value, i.e. given i, $Z^{(1)}(i)$ computes the representation $\boldsymbol{z}_i^{(1)}$ Similarly $Z^{(2)}$:

$$Z^{(2)} : [\![1 \,..\, M]\!] \to \mathbb{R}^{\ell_2}$$

$$u \mapsto \boldsymbol{z}_u^{(2)}$$

will handle the second factor such that $Z^{(2)}(j)$ computes $\boldsymbol{z}_j^{(2)}$.

In this case, there are a limited number of possible representations ($N + M$) as for each context class, there is only one representation. In this setting, it is impossible to have new classes at inference time (and therefore impossible to tackle our second task) because the encoders were trained on a set of values that cannot be expanded. Moreover, the latent spaces of contexts have no reason to be regular or well structured because only a few discrete values are used.

3.2.2.2 Decoder

Our decoder d takes as input the disentangled representation of each factor ($\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$) and outputs a reconstructed time series ($\hat{\mathbf{x}}_{i,j}$). It can either perform forecasting (if there is at least partial data for each time series) or generation (if no data is available for the time series). The formulation of d is quite simple for generation and a bit more complex for forecasting.

For generation, let us consider :

$$d : \mathbb{R}^{\ell_1} \times \mathbb{R}^{\ell_2} \to \mathbb{R}^T \tag{3.3}$$

$$\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)} \mapsto \hat{\mathbf{x}}_{i,j} \tag{3.4}$$

Any series $\mathbf{x}_{i,j}$ can be inferred by computing $d(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)})$. d is a parametrised model that can be represented by a neural network. If the series are of fixed length, the decoder can be Multi-Layer Perceptron (MLP), Convolutional Neural Network (ConvNet) or Recurrent Neural Network (RNN), if they have varying length, then only RNN can be used.

If we want to perform forecasting, our decoder must be a RNN. To leverage the (possibly partial) data that we have, our decoder must be able to take it as input. Only RNNs do not require the entire input in one batch. The time series will be reconstructed step by step, and for each step, the previous output (or the corresponding ground truth timestep) has to be given to our decoder. We denote $x_{i,j,t}$ the t^{th} time step of a series ($t \in [0, T-1]$)

In this case, let us consider :

$$\begin{aligned} d : [\mathbb{R}^{\ell_1} \times \mathbb{R}^{\ell_2}] \times \mathbb{R} &\to \mathbb{R} \\ \boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)}, \boldsymbol{x}_{i,j,t} &\mapsto \hat{x}_{i,j,(t+1)} \end{aligned}$$

Any series $\mathbf{x}_{i,j}$ can be inferred step by step by computing, for each $t \in [1, T-2]$: $\hat{x}_{i,j,t+1} = d(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)}, x_{i,j,t})$ and replacing $x_{i,j,t}$ by $\hat{x}_{i,j,t}$ when $x_{i,j,t}$ is not available.

These two settings allow us to perform either generation of forecasting without much change to the model or the training procedure.

3.2.2.3 Training procedure

As explained above, to reconstruct $\hat{\mathbf{x}}_{i,j}$ such that $m_{i,j} = 1$ (so that the series is in the training set) we simply chain our encoders with our decoder :

$$\hat{\mathbf{x}}_{i,j} = d\left(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)}\right) = d\left(Z^{(1)}(i), Z^{(2)}(j)\right) \tag{3.5}$$

The principle of the learning method is to minimise the loss between the ground truth and the reconstructed time series. This loss can be computed through the following equation.

$$\mathcal{L}\left(\mathbf{x}_{i,j}, d, Z^{(1)}, Z^{(2)}\right) = \Delta\left(\mathbf{x}_{i,j}, d\left(Z^{(1)}(i), Z^{(2)}(j)\right)\right) \tag{3.6}$$

where Δ is a loss function measuring the distance between the two series, such as Mean-Squared Error (MSE) . Our model is learned by minimising the average of this loss over all the examples from the training set. The final functions $d^*, Z^{(1)*}$ and $Z^{(2)*}$ are thus obtained by solving the following equation.

$$d^*, Z^{(1)*}, Z^{(2)*} = \underset{d, Z^{(1)}, Z^{(2)}}{\arg\min} \frac{1}{|\mathcal{N}|} \sum_{i \in N, j \in M, s.t. m_{i,j}=1} \Delta\left(\mathbf{x}_{i,j}, d\left(Z^{(1)}(i), Z^{(2)}(j)\right)\right) \tag{3.7}$$

where $\mathcal{N}$ is the set of examples in the training set ($|\mathcal{N}| = \sum m_{i,j}$). The encoders and decoder are learned jointly (end-to-end training).

When considering that d is a differentiable parameterised function (e.g. neural networks) and that $Z^{(1)}$ and $Z^{(2)}$ are matrices of embeddings, this problem can be solved by classical stochastic gradient descent techniques and back propagation.

The loss Δ we used throughout this chapter is the MSE, defined as :

$$\Delta(\hat{\mathbf{x}}_{i,j}, \mathbf{x}_{i,j}) = (\hat{\mathbf{x}}_{i,j} - \mathbf{x}_{i,j})^2 \tag{3.8}$$

3.2.2.4 Inference

At inference time, we want to estimate a new series $\mathbf{x}_{i,j}$ ($m_{i,j} = 0$), considering that there exist at least two series $\mathbf{x}_{i,k}$ and $\mathbf{x}_{k',j}$ that are observed ($m_{i,k} = m_{k',j} = 1$). This prediction is straightforward :

$$\hat{\mathbf{x}}_{i,j} = d(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)}) = d\left(Z^{(1)}(i), Z^{(2)}(j)\right) \tag{3.9}$$

As explained above, matrix completion is the only task this model can perform, because the number of representations the encoder can learn is fixed before training. To perform prediction on new context, the encoders have to be altered. The balancing advantage is that this system is very light and easy to adjust.

3.2.3 Inductive Model

This version of our model, described in Figure 3.4, can tackle both our tasks. This model is also comprised of an encoder and a decoder. While the decoder is the same as in the previous version, the encoders are now parametrised functions that take as input time series and not integers. To ensure disentanglement of the factor of variation, a specific training procedure has to be used. This version also adds an aggregation challenge at inference time. We first describe the encoder and decoder, then the modified training procedure and we finish with our solutions for inference.

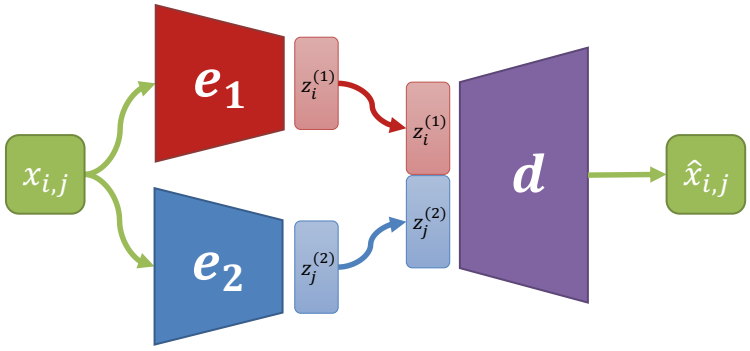

FIGURE 3.4 – The inductive setting, where the encoders are parametrised, taking time series as input, leading to several representations of a factor class.

3.2.3.1 Encoder

In this version, the encoders, are parametrised functions that take time series as input. Let us consider e_1 and e_2,

$$\begin{aligned} e_1 : \mathbb{R}^T &\to \mathbb{R}^{\ell_1} \\ \hat{\mathbf{x}}_{i,j} &\mapsto \boldsymbol{z}_i^{(1)} \\ e_2 : \mathbb{R}^T &\to \mathbb{R}^{\ell_2} \\ \hat{\mathbf{x}}_{i,j} &\mapsto \boldsymbol{z}_j^{(2)} \end{aligned}$$

e_1 is a function that, given any time series, is able to compute the representation of the first factor of this series i.e. given a series $\mathbf{x}_{i,j}$, $e_1(\mathbf{x}_{i,j})$ computes the representation $\boldsymbol{z}_i^{(1)}$. Similarly, e_2 will handle the second factor such that $e_2(\mathbf{x}_{i,j})$ computes $\boldsymbol{z}_j^{(2)}$.

The advantage of the version is that, given the continuous nature of the input space, we can encode new factor class **after** training, which is a requirement to tackle our second task.

The drawback is that, in this version, there is one representation of each class factor by example of this factor class in the dataset. To obtain a representation of the factor class i for the first factor (i.e. $\boldsymbol{z}_i^{(1)}$), we can encode $\mathbf{x}_{i,k}$ for any $k \in M$ and there is no reason why they should be identical. Formally, the following equation is very likely to be true :

$$\forall k, k' \in M \qquad k \neq k' \qquad e_1(\mathbf{x}_{i,k}) \neq e_1(\mathbf{x}_{i,k'})$$

and therefore there is no longer only one representation of each factor class (i.e. no unicity of $\boldsymbol{z}_i^{(1)}$). This presents a challenge during inference, and we present our solution in sous-sous-section 3.2.3.4.

3.2.3.2 Decoder

Our decoder is the same as previously (see sous-sous-section 3.2.2.2). It can either perform generation or forecasting with the same formulations and limitations as before.

3.2.3.3 Training Procedure

Learning d, e_1 and e_2 is not trivial. It typically cannot be achieved through a classical auto-encoding loss applied on each individual training series like the previous model. Indeed, in that case, the model would be unable to disentangle which information comes from the first factor of variation, and which information comes from the second one. In all likelihood, information pertaining to both factors of variation would be in both embeddings. We thus propose to rely on a learning objective that have been proposed in the disentanglement literature by **DentonB17**.

The principle of the learning method is to minimise the loss between a predicted series by using embeddings values computed from other series as illustrated in Figure 3.5.

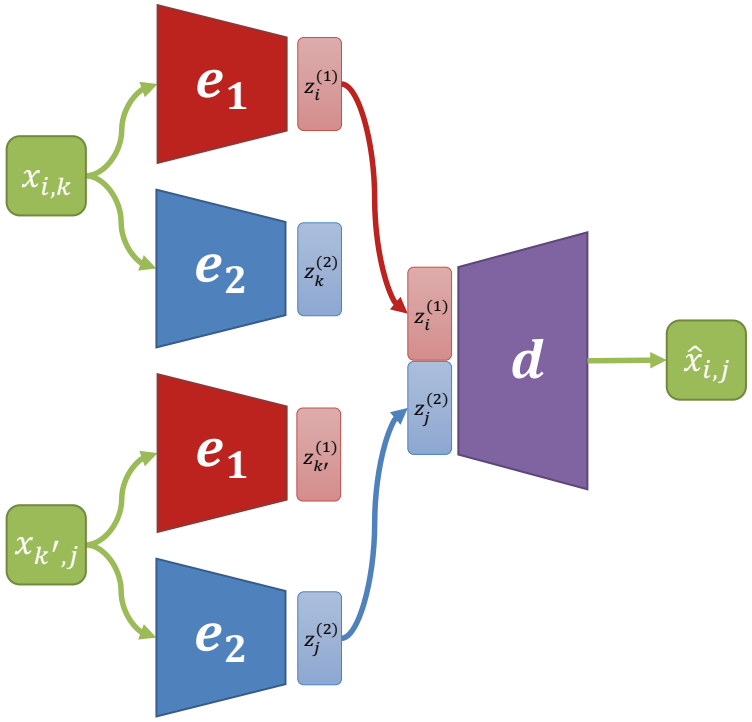

FIGURE 3.5 – The specific training procedure to ensure disentanglement in our inductive model. Using the inductive model as it is there would be no guarantee that the encoders would only encode the information pertaining to the factor we want them to encode. Therefore we have to use a specific training procedure, presented here, where the representations are mixed

Using the same notations as in the previous version of the model. Let us consider a triplet $\tau = (\mathbf{x}_{i,j}, \mathbf{x}_{i,k}, \mathbf{x}_{k',j})$ containing only series observed at train time i.e. $m_{i,j} = m_{i,k} = m_{k',j} = 1$. Given this triplet, we can compute the loss function.

$$\mathcal{L}(\tau, d, e_1, e_2) = \Delta(d(e_1(\mathbf{x}_{i,k}), e_2(\mathbf{x}_{k',j})), \mathbf{x}_{i,j}) \tag{3.10}$$

Our model is learned by minimising the average of this loss over all the triplets that can be constructed from the training set. The final functions d^*, e_1^* and e_2^* are thus obtained by solving this $\arg\min$:

$$d^*, e_1^*, e_2^* = \underset{d,e_1,e_2}{\arg\min} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k},\mathbf{x}_{k',j})\in\mathcal{T}} \Delta(d(e_1(\mathbf{x}_{i,k}), e_2(\mathbf{x}_{k',j})), \mathbf{x}_{i,j}) \tag{3.11}$$

where $\mathcal{T}$ is the set of all triplets that can be built from the training set.

In practice, we do not iterate over all the possible triplets. To improve computation time, for each example in one epoch only one triplet is randomly sampled.

Once again, d, e_1 and e_2 are differentiable parameterised functions (e.g. neural networks), so this problem can be solved by classical stochastic gradient descent techniques and back propagation.

3.2.3.4 Inference

Just as before, at test time, a series $\mathbf{x}_{i,j}$ can be predicted through $\mathbf{x}_{i,j} = d(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)})$, provided that some series $\mathbf{x}_{i,k}$ and $\mathbf{x}_{k',j}$ are observed to obtain $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$.

$$\boldsymbol{z}_i^{(1)} = e_1(\mathbf{x}_{i,k})$$
$$\boldsymbol{z}_j^{(2)} = e_2(\mathbf{x}_{k',j})$$

However, it may happen that multiple series $\mathbf{x}_{i,k}$ are observed for different values of k (and respectively for k' and $\mathbf{x}_{k',j}$). Specifically, in our first task, for both factors there are several examples of each factor class in the training set. In our second task, only one of the factor classes has several representants (the other one being seen for the first time has only one representant). This aggregation problem is illustrated in Figure 3.6

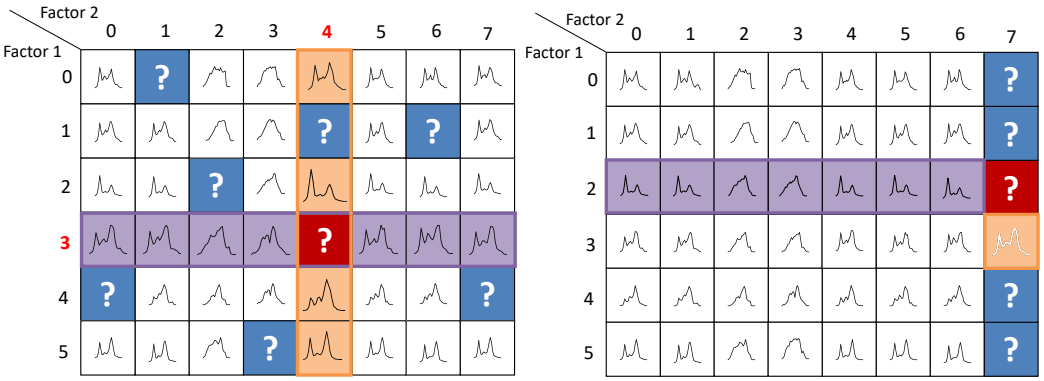

FIGURE 3.6 – In the first setting (on the left), there are several series that match the factor 1 index k and the factor 2 index k' (here $k = 3$ and $k' = 4$). To infer the series in red (index $3, 4$), the question is from which of them to obtain a representation of the two factors.
In the second setting (on the right) only one series (newly seen at inference) matches the factor 2 index k' but there are several series that matches the factor 1 index k (here $k = 2$ and $k' = 7$). The question remains from which series to obtain a representation of the factor 1 index k.

We propose two inference methods to obtain unique $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$, when they could be obtained from more than one example : one that averages in the latent space, the other one in the space of time series $\mathbb{R}^T$.

Embedding averaging : In the first method, the predicted series are computed by averaging at the embedding levels. We average the values of $e_1(\mathbf{x}_{i,k})$ and $e_2(\mathbf{x}_{k',j})$ on the

observed series, and use these vectors as inputs to the decoding function d. We denote $\mathcal{N}$ the subset of sequences' indices in the training set corresponding to the targeted factor 1 class i. We denote $\mathcal{M}$ the subset of sequences' indices in the training set corresponding to the targeted factor 2 class j. To average embedding in the latent space :

$$\boldsymbol{z}_i^{(1)\star} = \frac{1}{|\mathcal{N}|}\sum_{k\in\mathcal{N}} e_1(\mathbf{x}_{i,k}) \qquad \boldsymbol{z}_j^{(2)\star} = \frac{1}{|\mathcal{M}|}\sum_{k\in\mathcal{M}} e_2(\mathbf{x}_{k,j}),$$
$$\hat{\mathbf{x}}_{i,j} = d(\boldsymbol{z}_i^{(1)\star}, \boldsymbol{z}_j^{(2)\star})$$

Series averaging : The second method predicts the series as the average of the series predicted by using all the combinations of observed $\mathbf{x}_{i,k}$ and $bx_{k',j}$. With the same notation as above for $\mathcal{N}$ and $\mathcal{M}$ to average in the space of time series we use :

$$\hat{\mathbf{x}}_{i,j} = \frac{1}{|\mathcal{N}| + |\mathcal{M}|}\sum_{k\in N, k'\in M} d(e_1(\mathbf{x}_{i,k}), e_2(\mathbf{x}_{k',j}))$$

These two methods have their advantages and drawbacks. The first method forces the latent space to have some structure. Taking an average of the embeddings means that distance has a meaning in the space. However, we found that if we wanted to add a constraint on this space (such as making the encoded representation fit a Gaussian), this distance becomes less relevant. The second method does not add constraints to the space of representation, but is rather computationally intensive.

This aggregation challenge can also be solved with an attention mechanism which is the principle behind the third iteration of our model.

3.2.4 Model with attention

This version of our model, described in Figure 3.7, adds an attention mechanism to the previous model in order to have a unique representant for each factor class. The addition of the mechanism makes this model less disentangling than the previous ones. Indeed, the query required by the mechanism may contain information about the other factor. However, it also allows for more stability, especially when the training data is noisy.

3.2.4.1 Encoder

The encoders take the same form as in the inductive model. In order to emphasise the fact that there is no unique representation of a factor class, we denote $\boldsymbol{z}_{i,j}^{(1)}$ the representation of the factor 1 class i coming from $\mathbf{x}_{i,j}$ (instead of $\boldsymbol{z}_i^{(1)}$).

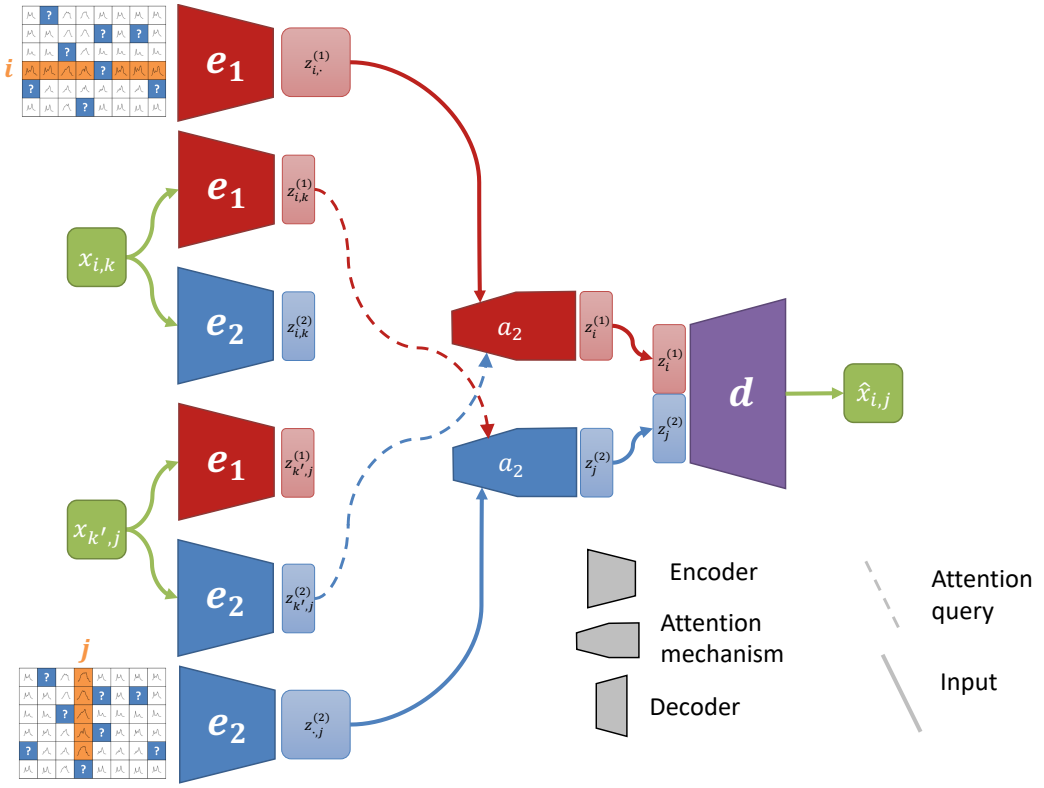

FIGURE 3.7 – The setting with attention, where an attention mechanism allows for learning an optimal representant for each factor class.

Let us consider e_1 and e_2,

$$\begin{aligned} e_1 : \mathbb{R}^T &\to \mathbb{R}^{\ell_1} \\ \mathbf{x}_{i,j} &\mapsto \boldsymbol{z}^{(1)}_{i,j} \\ e_2 : \mathbb{R}^T &\to \mathbb{R}^{\ell_2} \\ \mathbf{x}_{i,j} &\mapsto \boldsymbol{z}^{(2)}_{i,j} \end{aligned}$$

e_1 is a function that, given any time series is able to compute the representation of the first factor of this specific series. Given a series $\mathbf{x}_{i,j}$, $e_1(\mathbf{x}_{i,j})$ computes the representation $\boldsymbol{z}^{(1)}_{i,j}$. Similarly, e_2 will handle the second factor such that $e_2(\mathbf{x}_{i,j})$ computes $\boldsymbol{z}^{(2)}_{i,j}$.

3.2.4.2 Attention Mechanism

We added the mechanism to obtain a unique representation of each factor class as input for the decoder. Like the encoders, we need two attention mechanisms (one by factor). With the mechanism, the representation $\boldsymbol{z}^{(1)}_i$ is based on the aggregation of all the possible representations of $\boldsymbol{z}^{(1)}_{i,\cdot}$ that can be computed from the training set with respect to a query : $\boldsymbol{z}^{(2)}_{i,j}$. The notation $\boldsymbol{z}^{(1)}_{i,\cdot}$ corresponds to the set of all encoded examples from the

training set that have i as factor 1 class. We denote $\mathcal{M}$ the set of corresponding indices. Our attention mechanism will compute a weight for each example in $\boldsymbol{z}_{i,\cdot}^{(1)}$ and output the weighted sum.

Let us consider a_1 and a_2 two attention functions :

$$\begin{aligned} a_1 : \mathbb{R}^{\ell_1 \times \mathcal{N}} \times \mathbb{R}^{\ell_2} &\to \mathbb{R}^{\ell_1} \\ \boldsymbol{z}_{i,\cdot}^{(1)}, \boldsymbol{z}_{i,j}^{(2)} &\mapsto \boldsymbol{z}_i^{(1)} \\ a_2 : \mathbb{R}^{\ell_2 \times \mathcal{M}} \times \mathbb{R}^{\ell_1} &\to \mathbb{R}^{\ell_2} \\ \boldsymbol{z}_{\cdot,j}^{(2)}, \boldsymbol{z}_{i,j}^{(1)} &\mapsto \boldsymbol{z}_j^{(2)} \end{aligned}$$

a_1 is a function that, given a query $\boldsymbol{z}_{i,j}^{(2)}$ and a set of representations $\boldsymbol{z}_{i,\cdot}^{(1)}$, will compute $\boldsymbol{z}_i^{(1)}$, the representation of the factor 1 class i. It is worth noting that this notation is not quite exact, since $\boldsymbol{z}_i^{(1)}$ will depend on the query and not just on the factor class i.

$$\boldsymbol{z}_{i,\cdot}^{(1)} = \{e_1(\mathbf{x}_{i,k})\} \qquad \text{for } k \in \mathcal{M} \tag{3.12}$$

Once we have $\boldsymbol{z}_{i,\cdot}^{(1)}$, to compute this representation, we first compute via scalar product w a vector of weights and then perform the weighted sum.

$$\boldsymbol{z}_{i,j}^{(2)} = e_2(\mathbf{x}_{i,j}) \tag{3.13}$$
$$w_k = \boldsymbol{z}_{i,k}^{(1)} \cdot A_{1\to 2} \cdot \boldsymbol{z}_{i,j}^{(2)}) \tag{3.14}$$
$$\boldsymbol{z}_i^{(1)} = a_1(i, \boldsymbol{z}_{i,j}^{(2)}) = \sum_k w_k \boldsymbol{z}_{i,k}^{(1)} \tag{3.15}$$

Similarly, a_2 will compute $\boldsymbol{z}_j^{(2)}$ from a set of representation $\boldsymbol{z}_{\cdot,j}^{(2)}$ and a query $\boldsymbol{z}_{i,j}^{(1)}$

$$\boldsymbol{z}_{i,j}^{(1)} = e_1(\mathbf{x}_{i,j}) \tag{3.16}$$
$$w_k = \boldsymbol{z}_{k,j}^{(2)} \cdot A_{2\to 1} \cdot \boldsymbol{z}_{i,j}^{(1)} \tag{3.17}$$
$$\boldsymbol{z}_j^{(2)} = a_2(j, \boldsymbol{z}_{i,j}^{(1)}) = \sum_k w_k \boldsymbol{z}_{k,j}^{(2)} \tag{3.18}$$

$A_{1\to 2}$ and $A_{2\to 1}$ are matrices that are needed because ℓ_1 and ℓ_2 can be different. They allow for the projection of a representation of one factor into the space of the representation of the other factor. They both have $\ell_1 \times \ell_2$ parameters that are learned during the training procedure.

For the sake of clarity, we denote $\boldsymbol{z}_i^{(1)} = a_1(i, \boldsymbol{z}_{i,j}^{(2)})$ instead of $a_1(\boldsymbol{z}_{i,\cdot}^{(1)}, \boldsymbol{z}_{i,j}^{(2)})$.

According to our experiments, the attention functions should measure the interest of $\boldsymbol{z}_{i,\cdot}^{(1)}$ with respect to the query $\boldsymbol{z}_{i,j}^{(2)}$, the second factor associated with $\mathbf{x}_{i,j}$.

Figure 3.8 illustrates the output of our attention mechanism for an example. All the representations of a factor class have been assigned a weight (shown in levels of grey).

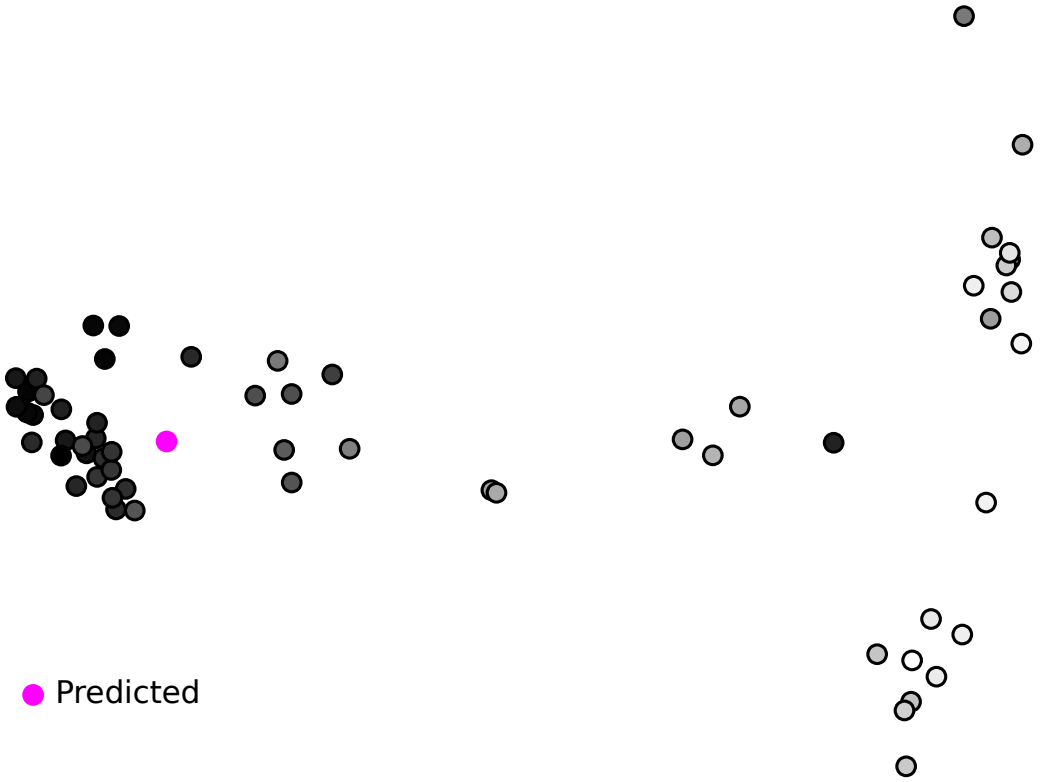

FIGURE 3.8 – The PCA in 2 dimensions of the result of our attention mechanism for the latent space of one factor in one of our datasets. The mechanism assigns each representation a weight (shown in levels of grey) and the weighted sum gives us the predicted value of a factor's representation (for a specific query).

In addition to solving our aggregation problem, this mechanism makes the model more stable. It is more robust to noisy data (such as mislabeled examples), because the model will learn to assign a small weight to the misclassified examples.

3.2.4.3 Decoder

Our decoder is the same as previously (see sous-sous-section 3.2.2.2). It can either perform generation or forecasting with the same formulations and limitations as before.

3.2.4.4 Training Procedure

The training procedure is similar to the previous one, with the addition of our attention mechanism. The inputs are given to the encoders, the outputed representations are then passed through the attention mechanism. The results of the query in the attention mechanism are passed in a mixed manner to the decoder.

We use the same notations as in the previous version of the model. Let us consider a triplet $\tau = (\mathbf{x}_{i,j}, \mathbf{x}_{i,k}, \mathbf{x}_{k',j})$ containing only series observed at train time i.e. $m_{i,j} = m_{i,k} = m_{k',j} = 1$. Given this triplet, we can compute the loss function.

$$\mathcal{L}(\tau, d, e_1, e_2) = \Delta(d(a_1(i, e_2(\mathbf{x}_{i,k})), a_2(j, e_1(\mathbf{x}_{k',j}))), \mathbf{x}_{i,j})$$

Once again, our model is learned by minimising the average this loss over all the triplets that can be constructed from the training set. The final functions d^*, e_1^*, e_2^*, a_1^* and a_2^* are thus obtained by solving the following $\arg\min$:

$$d^*, e_1^*, e_2^*, a_1^*, a_2^* = \underset{d,e_1,e_2,a_1,a_2}{\arg\min} \frac{1}{|\mathcal{T}|} \sum_{(\mathbf{x}_{i,j},\mathbf{x}_{i,k},\mathbf{x}_{k',j})\in\mathcal{T}} \Delta\left(d(a_1(i, e_2(\mathbf{x}_{i,k})), a_2(j, e_1(\mathbf{x}_{k',j}))), \mathbf{x}_{i,j}\right) \tag{3.19}$$

where $\mathcal{T}$ is the set of all triplets that can be built from the training set. In practice, we do not iterate on all the possible triplets. To improve computation time, for each example in one epoch, only one triplet is randomly sampled. We also used an implementation trick inspired by adversarial techniques such as Generative Adversarial Network (GAN), where for each mini-batch, all examples have one identical factor class.

Once again, d, e_1, e_2, a_1 and a_2 are differentiable parameterised functions (e.g. neural networks), so this problem can be solved by classical stochastic gradient descent techniques and back propagation.

3.2.4.5 Inference

Just as before, at test time, a series $\mathbf{x}_{i,j}$ can be predicted through $\mathbf{x}_{i,j} = d(\boldsymbol{z}_i^{(1)}, \boldsymbol{z}_j^{(2)})$, provided that some series $\mathbf{x}_{i,k}$ and $\mathbf{x}_{k',j}$ are observed to obtain $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_j^{(2)}$. Our aggregation problem is now different. We only need to choose a query to use with our attention function.

For our first task, we have multiple series that can be used to compute both factor's representation and therefore don't have a defined query. We decided to use an average of the possible representations for each factor. Similarly to the previous inductive model :

$$\boldsymbol{z}_i^{(1)\star} = \frac{1}{|\mathcal{N}|} \sum_{k\in\mathcal{N}} e_1(\mathbf{x}_{i,k}) \qquad \boldsymbol{z}_j^{(2)\star} = \frac{1}{|\mathcal{M}|} \sum_{k\in\mathcal{M}} e_2(\mathbf{x}_{k,j})$$

$$\hat{\mathbf{x}}_{i,j} = d(a_1(i, \boldsymbol{z}_j^{(2)\star}), a_2(j, \boldsymbol{z}_i^{(1)\star}))$$

For our second task, the process is straightforward. We have an example with an unobserved factor class (for example $\mathbf{x}_{i,\hat{k}}$ where the new factor class is denoted $\hat{k}$). We

wish to compute the set of $\mathbf{x}_{\cdot,\hat{k}}$. For any factor 2 class u observed in the training set, we compute

$$\hat{\mathbf{x}}_{u,\hat{k}} = d(a_1(u, e_2(\mathbf{x}_{i,\hat{k}})), e_2(\mathbf{x}_{i,\hat{k}}))$$

This concludes the presentation of our models. We have three declinations, each adapted to specific situations. Our first version, transductive, is designed to perform our first task, matrix completion. The second version, inductive, can perform both tasks. It unfortunately lacks a unique representation of the factor. For one factor class, there are as many representations of this factor than there are examples in the dataset that match this factor. Our third version, with an attention mechanism, learns how to obtain a unique representation for each factor. It is mostly aimed at the second task : new factor class. It is also more stable and less sensitive to noise, as our experiments will show.

3.3 Experiments

Our models were tested in two different settings (clean or noisy labels) with four different datasets. We compared our results to different baselines. sous-section 3.3.1 presents the four datasets and the two different settings (noisy or not) we experimented on. sous-section 3.3.2 presents the baseline we compared our models to and sous-section 3.3.3 presents the quantitative results of our different experimentations. Finally, sous-section 3.3.4 shows a few qualitative results on the learned latent spaces.

3.3.1 Datasets

We tested our models on four datasets organised along two factors of variation. In each case, the first factor of variation is the time (each factor class is a day) and the second one is a location (each factor class corresponds to a spatial context). Three of these datasets are publicly available. The first dataset consists of public transportation system information and is not public. The second dataset is public and relates to energy consumption in regions of the north-east of the USA. Our third and fourth datasets were extracted from a single public dataset that relates to air quality and pollutants in the city of Madrid.

3.3.1.1 STIF

The first dataset, named STIF, is based on transportation data provided by Ile-de-France Mobilites (formerly STIF, the transport agency of the Parisian region)[3]. The dataset consists of all the logs between October and December 2015 in the bus, subway, trams and RER (suburb trains) networks along with their time-stamps. Each line in the dataset is a tuple (u, s, t) where u is the user id, s the tap-in location and t the time-stamp. We decided to focus on the subway network composed of 300 stations.

3. https ://www.iledefrance-mobilites.fr/

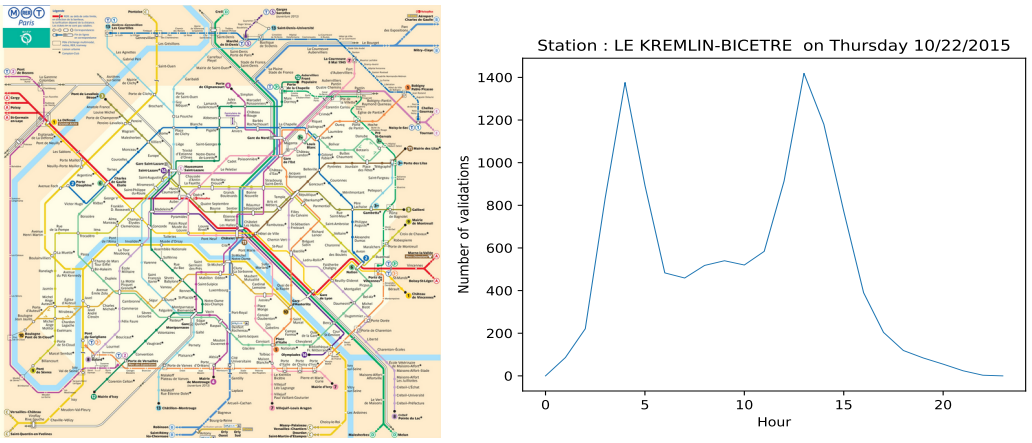

FIGURE 3.9 – The STIF dataset. On the left, a map of the subway network of Paris. On the right, one example of a time series from the STIF dataset. This dataset is comprised of 297 stations and 89 days.

Transportation data are inherently discontinuous because the subway is closed from approximately 2 AM to 5 AM (depending on the day of the week). As the closing times are different from one station to another and from one day to another, we have chosen to keep a 24-hour format with a few null time step. Thus, we consider one series per 24h per station, each series containing 24 data points. From the dataset we remove 3 stations which were undergoing planned works during the period. We also removed 3 anomalous days : the 14^{th} of November the day after the Paris terrorist attacks; the 29^{th} and 30^{th} of November which were the opening days of COP 21 where the subway was free and therefore no validations were made.

Our data is processed in the form of a 3D tensor X of shape $D \times N \times T$ where D (89), N (297) and T (24) are respectively the number of days, stations and time steps. In other words, $\mathbf{x}_{i,j}$ corresponds to the i^{th} row and j^{th} column of X.

3.3.1.2 Energy

The second dataset, called Energy, is from Kaggle [4]. It consists of hourly consumption of energy in the East of the United States. It comes from PJM Interconnection LLC [5], a regional transmission organisation (RTO) in the United States that supplies energy to a large portion of the North-East of the US. The energy grid is split over 10 different regions, each of them reporting consumption only over the area it covers.

The data spans from the 2^{nd} of June 2013 to the 2^{nd} of August 2018. A few days had recording errors and were dismissed. Our data is processed in the form of a 3D tensor X of shape $D \times N \times T$ where D (1877), N (10) and T (24) are respectively the number of days, regions and time steps. In other words, $\mathbf{x}_{i,j}$ correspond to the i^{th} row and j^{th} column of X.

4. https ://www.kaggle.com/robikscube/hourly-energy-consumption/
5. PJM : www.pjm.com

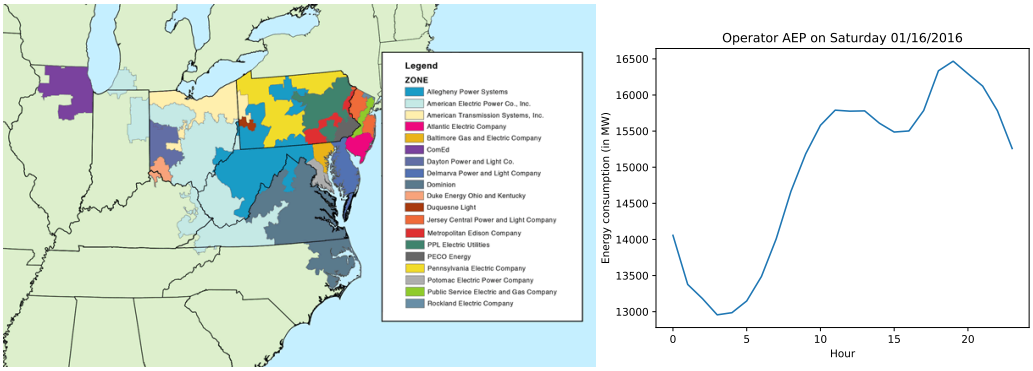

FIGURE 3.10 – The Energy dataset. On the left, a map of the region covered by PJM. On the right, one example of a time series from the energy dataset. This dataset is comprised of 10 stations and 1877 days

3.3.1.3 AirQuality

The third and fourth datasets also came from Kaggle [6] but were collected and opened by the City of Madrid [7]. This dataset consists of hourly measures of several pollutants in the air of Madrid. We focused on the pollutants measure by the maximum number of stations : NO_2 and NO. The hourly measures are reported by 24 stations in $\mu g/m^3$.

FIGURE 3.11 – The NO and NO_2 datasets. On the left, a map showing the position of the measurement stations. On the right, one example of two time series from the AirQuality datasets, one for NO the other for NO_2. We split the two pollutants into two datasets composed of 24 stations and 2669 days.

The measurements cover the period from the 2^{nd} of January 2011 to the 30^{th} of April 2018. A few days had missing values and were dismissed. We decided to split this dataset into two subdatasets, the first one comprised of the measurements of NO, the second of the measurements of $NO2$.

6. https ://www.kaggle.com/decide-soluciones/air-quality-madrid/
7. https ://datos.madrid.es/portal/site/egob

Our data is processed in the form of two 3D tensors X of shape $D \times N \times T$ where D (2669), N (24) and T (24) are respectively the number of days, stations and time steps. In other words, $\mathbf{x}_{i,j}$ correspond to the i^{th} row and j^{th} column of X.

3.3.1.4 Pre-processing

These datasets correspond to very different applications : public transportation traffic, energy consumption and air quality measurements.

However, they share common properties : stationarity for weekdays and specific operating modes for the weekends, following human activities. These features mean that there are very sensitive to contexts and it should be relatively easy to learn context rich enough so that they are mostly stationary with regard to them. It also means that the usual contextual baselines are very strong predictors and will be hard to overcome.

Indeed, for each of these datasets, the knowledge of the spatiotemporal contexts is very informative. They are all the results of the aggregation of daily human activity, which is quite regular. Even in the individual activities vary, they are also often correlated. The usual way to approximate them is to average on similar contexts.

All datasets were normalised between 0 and 1 via min-max scaling. Because they were inherently multi-scaled (for example some subway stations are much bigger than others and the most frequented parts of town have more pollutants than others), this normalisation was done by the factors corresponding to the spatial context. They were split into train, validation and test sets with a classical $70\%/15\%/15\%$ ratio.

3.3.2 Baselines

To evaluate the relevance of our models, we compared their performances to several baselines. Because our models are at the junction of dynamic and contextual approaches, we must explore both perspectives.

On the one side, the most classical way to perform an efficient forecasting is to rely on autoregressive models (see **??**) where $x_{i,j,t}$ is estimated by a weighted linear combination of its N previous observations $\mathbf{x}_{i,j,t-\tau}$ possibly supplemented by moving averages to fight noise [**sarima**]. This is a strong baseline for forecasting, based on the dynamic of the signal, but is known to worsen as the number of steps to forecast increases.

On the other side, another way to tackle time series forecasting consists in modelling only contextual information. In this case, we compute the average of all signals corresponding to a given context (i.e. a specific combination of external factors) and we assume that the resulting output will be a good predictor for the whole period (24 hours in our examples). We assume here two things. The first is that contexts are limited to the spatial and temporal contexts. The second is that these contexts are enough to encompass most of the information the time series do. Obviously, this assumption is only valid for certain applications ; however, we are convinced that this class of applications is rather large. For example, although pollution forecasts (and to a lesser extent energy consumption) are

strongly connected with weather models, we believe that spatial and temporal context will be enough. Public transportation predictors already rely heavily on contextual aggregations (such as the day of the week). Even in sales forecasts, state of the art approaches largely rely on context modelling as shown by **prophet**.

The Autoregressive (AR) approach is supposed to be very efficient at $t+1$, due to the fact that it is trained for this purpose. However, AR models are known to underperform on longer time scale. On the other side, we expect our contextual baseline to be very strong on a long-term basis, but, as it has no access to dynamic information, the short term would not be better.

We have developed four baselines. The first baseline is contextual and mainly aimed at our first task (matrix completion). The second is designed only for the second task (new context) and relies and clustering and contexts. The third is oracle (it needs access to the ground truth) and can be used in both tasks. The fourth baseline uses AR models.

3.3.2.1 Average Baseline

This baseline is considered for both tasks. It predicts $\hat{\mathbf{x}}_{i,j}$ by averaging the observed time series sharing a common factor i or j :

$$\hat{\mathbf{x}}_{i,j} = \frac{1}{|\mathcal{M}| + |\mathcal{N}|} \left(\sum_{k \in \mathcal{M}} \mathbf{x}_{i,k} + \sum_{k \in \mathcal{N}} \mathbf{x}_{k,j} \right) \tag{3.20}$$

where $\mathcal{N}$ (respectively $\mathcal{M}$) denotes the set of all series involving factor i (respectively factor j) in the training set. This baseline is illustrated in Figure 3.12.

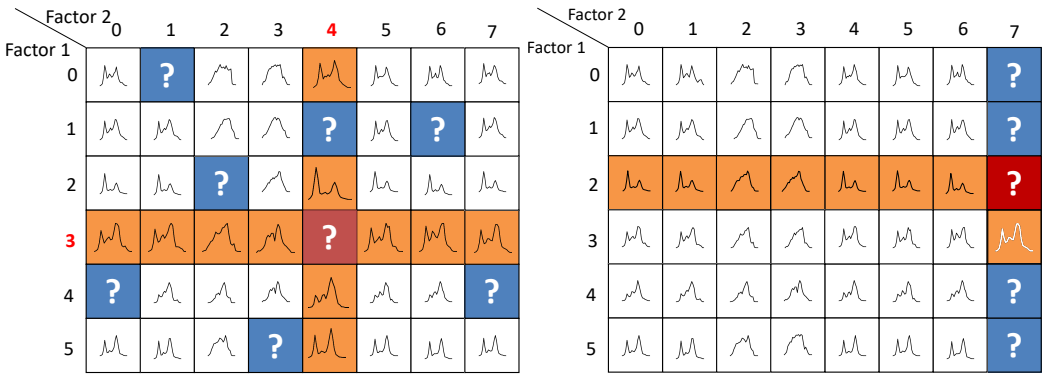

FIGURE 3.12 – The average baseline. For the first setting (on the left), there are many series that match each factor. For the second setting (on the right) for one of the factors, there is only one matching series. In both cases, all matching series (in orange) are averaged to predict the series in red.

This baseline can be used for both our tasks. The only difference is that for matrix completion both $|\mathcal{N}|$ and $|\mathcal{M}|$ are strictly greater than 1, while for the new factor value task, either $|\mathcal{N}|$ or $|\mathcal{M}|$ is equal to 1.

3.3.2.2 k-nearest neighbours baseline

This intuitive approach is dedicated to the 2^{nd} task (new factor class). In this task, we are given $\mathbf{x}_{i,\hat{k}}$,where $\hat{k}$ corresponds to a new factor 2 class while i corresponds to an observed factor 1 class. We aim at predicting all other $\mathbf{x}_{u,\hat{k}}$ for $u \in \mathcal{N}$, where $\mathcal{N}$ is the set of factor 1 class values observed in the training set.

The principle of the baseline is to find the p series in $\{\mathbf{x}_{i,\cdot}\}$ that are the closest to $\mathbf{x}_{i,\hat{k}}$ Let us define $\mathcal{U}$ the subset of factor 1 class values corresponding to these series. Then, $x_{\hat{i},\hat{k}}$ is predicted with the average of the p series $\{\mathbf{x}_{u,\hat{j}}\}_{u\in\mathcal{U}}$

Then our targeted series is estimated as :

$$\hat{\mathbf{x}}_{i,j} = \frac{1}{p}\sum_{k\in\mathcal{U}} \mathbf{x}_{k,j} \tag{3.21}$$

These equations do not reflect the simplicity of this baseline that is illustrated in Figure 3.13.

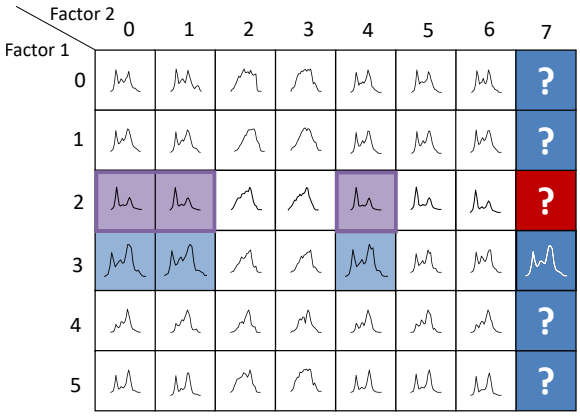

FIGURE 3.13 – The k-nearest neighbours baseline. Given a new series with a factor 1 index not seen during training (here index 7), we find its closest neighbours in the training set matching the index of the factor 2 (in light blue here index $0, 1, 4$). To infer another series (in red here, index $(2, 7)$), we use the indices of the factor 1 of the series previously found to infer what would be the closest neighbours of the series to infer (the predicted closest neighbours are in purple here). We then average our predicted closest neighbours for inference.

For this baseline, like for the next one, the number of neighbours p is an hyperparameters we optimise.

3.3.2.3 Oracle k-nearest neighbours baseline

The third baseline is an oracle since it requires access the ground truth. $\mathbf{x}_{i,j}$ is simply estimated by averaging its p nearest neighbours in the training set.

Given the stability and regularity of the datasets, $\mathbf{x}_{i,j}$ is likely to have at least an almost twin in the dataset and this oracle will be particularly strong.

3.3.2.4 Auto-regressive models

The hourly format of the data is intrinsically low noise. We rely on a basic N-order autoregressive model (see **??**) trained to minimise the Mean Squared Error loss (MSE) [8].

$$\hat{\mathbf{x}}_{i,j}[t] = f_{AR}(\mathbf{x}_{i,j}[:t]) = \alpha_0 + \sum_{\tau=1}^{N} \alpha_\tau x_{i,j}[t-\tau], \qquad \mathcal{L} = \sum_t (f_{AR}(\mathbf{x}_{i,j}[:t]) - \mathbf{x}_{i,j}[t])^2 \quad (3.22)$$

Classically, the trained model can be fed with $\hat{\mathbf{x}}_{i,j}[t]$ to predict $\hat{\mathbf{x}}_{i,j}[t+1]$.

3.3.3 Results

We first give our results for both tasks on data without noise. We then give our results on both tasks with artificially noised data to evaluate the resilience of our models.

3.3.3.1 Original data without noise, task 1 : matrix completion

Tableau 3.1 shows our results for our first task (matrix completion) with the MSE between the predicted series and the ground truth. As expected, the oracle baseline achieves near perfect results. The setting with embeddings matrices often outperforms the neural network settings. We believe it is because it encodes the exact information of the factor class values, whereas the neural network settings do not have access to them. The model using the attention mechanism also outperforms the other settings using neural networks. It also either outperforms the transductive setting or comes very close. The use of the attention mechanism instead of the other aggregations methods allows gain from 26% up to 48%, proving that it is an improvement of our model.

3.3.3.2 Original data without noise, task 2 : new context value

For our second task, we want to make prediction for all factors 2 from a single sequence associated with a new factor 1 (or respectively extrapolating predictions for all factor 1 from one sequence corresponding to a new factor 2).

Tableau 3.2 shows our results on this task (same MSE as the previous one). Except for the results on the new factor 2 (location) for the datasets on Energy and Air quality, our models outperform our baselines. We believe this is due to the small number of factors 2 in these two datasets (10 and 24). Once again, the model with attention mechanism is a clear improvement on the other ones with gain that range from 10 % to 30%

8. We use the python like syntax $\mathbf{x}_{i,j}[:t]$ to denote the sequence of values. $[x_1, \ldots, x_{t-1}]$

		STIF			Energy			NO			NO2		
Encoder	Decoder	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn
Oracle NN (p)		1 (5)			0.8 (4)			7 (5)			31 (5)		
BL avg (temporal)		179			60			107			215		
BL avg (spatial)		121			89			45			98		
Emb	MLP	28			20			43			**98**		
Emb	CNN	53			48			46			101		
Emb	RNN	**16**			**15**			**41**			102		
MLP	MLP	84	84	62	46	47	32	52	52	39	106	105	93
CNN	CNN	73	73	51	59	59	38	58	57	**35**	104	105	92
RNN	RNN	26	25	**18**	36	34	**18**	50	51	36	107	105	**83**

TABLE 3.1 – MSE errors for task 1 : sequence generation with the matrix completion setting. From top to bottom : baselines, transductive modelling & inductive modelling (context encoded on the fly). For sake of clarity, all errors have been multiplied by 10000.

			STIF			Energy			NO			NO2		
	Encoder	Decoder	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn
New loc.	BL NN (p)		123 (150)			81 (3)			49 (5)			107 (5)		
	BL avg.		125			87			52			112		
	MLP	MLP	59	59	56	118	118	80	58	57	52	107	107	106
	CNN	CNN	75	76	72	99	102	96	62	64	66	101	102	100
	RNN	RNN	49	49	**44**	107	108	**70**	62	65	**53**	106	108	**99**
New day	BL NN		143 (7)			93 (900)			99 (70)			207 (60)		
	BL avg.		145			89			103			210		
	MLP	MLP	63	61	46	76	76	69	62	62	63	133	133	131
	CNN	CNN	61	59	52	65	63	40	71	71	66	129	128	122
	RNN	RNN	41	40	**29**	47	46	**19**	63	63	**55**	139	139	**112**

TABLE 3.2 – MSE errors for task 2 : generation of series associated with a new factor. New locations above, new days below. For sake of clarity, all errors have been multiplied by 10,000.

3.3.3.3 Data with noise, task 1 : matrix completion

To prove the efficiency of our attention mechanism with regard to noise, we introduced some in the datasets. To artificially add some noise, we swapped the label of some series in the training set, meaning that at least one of the factors classes does not match the real ground truth of the series. In :fig :task1@cref

:fig :task1@cref

:fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task1@cref

:fig :task2@cref

:fig :task2@cref

:fig :task1@cref

3,3,3,3,3 :fig :task2@cref

3,3,3,3,3 :fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task2@cref

:fig :task1@cref

:fig :task1@cref

?? this corresponds to swapping a fraction of the series with a white background. In practice, we randomly sampled 15% of the training set and cycled the labelled factor so that the examples would have at least one mislabeled factor. The choice of 15% was a bit arbitrary but seemed like a good balance between credible errors in real world datasets and too noisy to be usable. This section and the next present the result we obtained on these noisy datasets.

Tableau 3.3 shows our MSE results for our first task (matrix completion) on the noisy data. The oracle baseline, though a bit disturbed by the noise, still achieves very good results. All the other baselines, however, achieve lesser results than our models. Once again, the attention mechanism achieves better results than the other aggregation methods with gains ranging from 40% to 60%. The difference in the error with the previous un-noisy data shows that the settings without attention are comparatively more impacted in their results than the setting with attention. This difference in the degradation of the results range from 10 % to 200%. This show the strength and resilience the attention mechanism brings to the model.

		STIF			Energy			NO			NO2		
Encoder	Decoder	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn
Oracle NN (p)		11 (5)			1 (5)			13 (5)			44 (6)		
BL avg (temporal)		270			89			119			227		
BL avg (spatial)		171			99			98			214		
MLP	MLP	162	163	69	96	97	80	86	87	57	185	188	98
CNN	CNN	163	163	65	69	72	45	87	87	**44**	193	193	94
RNN	RNN	164	164	**59**	55	57	**32**	89	89	62	178	189	**91**

TABLE 3.3 – MSE errors for task 1 : sequence generation with the matrix completion setting with **noisy data**. From top to bottom : baselines, transductive modelling & inductive modelling (context encoded on the fly). For sake of clarity, all errors have been multiplied by 10,000.

3.3.3.4 Data with noise, task 2 : new context value

Tableau 3.4 shows our MSE results on this task for new factor values with the same 15% of swapped labels. Similarly to the setting when there is no noise, our models outperform the baselines, except for the NO_2 dataset on new location (factor 2) which we again explain with the small numbers of contexts. The models with attention mechanisms achieve better performance than the ones without and manage to limit the impact of the noise in the data though not as clearly as in the previous experiment.

			STIF			Energy			NO			NO2		
	Encoder	Decoder	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn	Series avg.	Emb. avg	Attn
New loc.	BL NN (p)		211 (130)			135 (3)			89 (6)			127 (5)		
	BL avg.		242			146			95			206		
	MLP	MLP	163	162	158	159	160	97	94	95	84	190	193	180
	CNN	CNN	164	165	158	102	102	90	90	90	**86**	185	187	**178**
	RNN	RNN	172	173	**159**	102	101	**89**	94	92	87	184	185	**178**
New day	BL NN		183 (5)			94 (900)			99 (70)			207 (60)		
	BL avg.		155			102			112			215		
	MLP	MLP	100	102	93	91	91	76	74	74	72	160	162	159
	CNN	CNN	98	99	98	65	66	58	74	74	73	163	163	158
	RNN	RNN	93	93	**89**	52	50	**42**	70	72	**66**	162	163	**140**

TABLE 3.4 – MSE errors for task 2 with noisy data : generation of series associated with a new factor. New locations above, new days below. For sake of clarity, all errors have been multiplied by 10,000.

3.3.4 Qualitative Results

This section explores the explicative power of our model, to show that the learned latent spaces are indeed meaningful. As explain in the chapitre 2, we want our latent spaces to be not only disentangled but also continuous and complete.

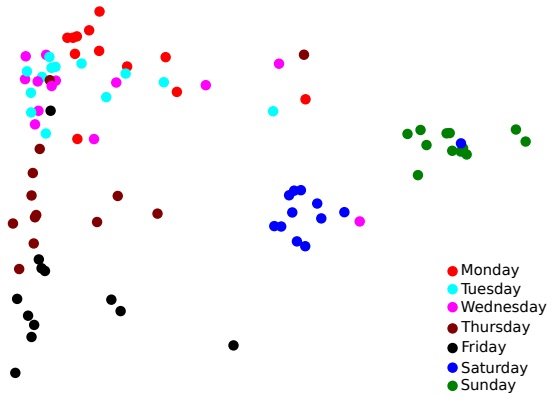

FIGURE 3.14 – PCA of the representations of the different day of the STIF dataset in their latent space, no attention, no noise. The pink dot in the cluster of blue represents the 11^{th} of November which is a holiday in France.

As an example, Figure 3.14 shows the Principal Component Analysis (PCA) of the set of representation of all days (factor 1) present in the STIF dataset. It shows that our model's latent space of representation has grouped together the representations corresponding to the same day of the week (information that is not present in the dataset). Moreover, the Saturdays and Sundays are clearly differentiated from the weekdays which can be explained by the habits of the regular users of the Parisian network that are different on the weekend than there are during the week.

For our model with an attention mechanism, Figure 3.15 shows the PCA of the representation of the days (factor 1) in the training set of the STIF dataset. We can see that the output of the attention mechanism is close to the ground truth. Unfortunately, the attention mechanism seems to negatively impact the structure of the latent space. Because the average of representation does not necessarily have a meaning in this model, the latent space may be less regular.

Finally, when we add noise to the dataset by mixing up some of the labels in the training set, we can see in Figure 3.16 that our model with attention is less disturbed by the noise and outputs a representation close to the ground truth. In both figures there are 15% of mislabeled examples whose representations (circles) are not aligned with their correctly labelled counterpart (squares). For the setting without attention (on the left), this results in the prediction with noise being more distant to the encoded value of the ground truth than the prediction without noise. For the setting with attention, the distance is much

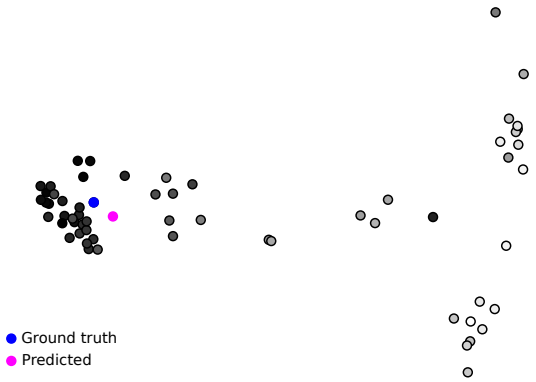

FIGURE 3.15 – PCA of the representation of the different days of the STIF training set, with attention, without noise. The level of grey indicates the weight of the attention. The blue dot is the representation of the ground truth encoded factor. The magenta dot represents the weighted sum that is the output of the attention mechanism. Without the attention mechanism, the aggregation method in the latent space would have average all the factors.

less important which tends to show that the attention mechanism reduces the sensitivity of the model to noise.

3.4 Conclusion

We developed a model (or a series of models) that can tackle two tasks by learning representations of contexts. The first task is a missing data setting in which data was only collected for some combinations of contexts. The goal is to recover the data for the other combinations. We believe this could help in many situations where collecting data is expensive. With this setting, collecting data only in some places and times could be enough to recover the rest. The second task is a bit different. We want to make predictions on contexts that were not seen during training. This means that we should be able to learn on the fly, not just during training. It is akin to one or few-shot learning. It could be particularly useful in domains such as collaborative filtering where the cold start problem is a great challenge. The ability to encode new data at inference time would allow for the addition of new products after training.

Our models are based on an encoder-decoder scheme. The encoder learns disentangled representation of contexts. The decoder decodes combinations of these representations to predict time series. The difference between our models largely lies with the encoder. Our first model uses embeddings matrices. This means that it cannot perform our second task, because the set of possible contexts is fixed after training. We showed that it can perform really well on our first task. Our second model learns disentangled representation of

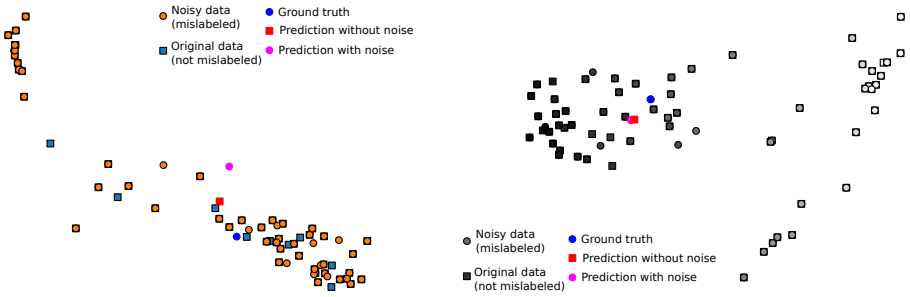

FIGURE 3.16 – Representations of the different days of the STIF training set with noise. The (blue) squares represent the original data, without noise while the (orange) dots represent the noisy data. The dark blue dot is the representation of the ground truth encoded factor. The magenta dot represents the representation given as input in the decoder. The red square is the representation that would have been given if there was no noise in the data. On the left without attention mechanism, on the right, with the attention mechanism and the level of grey indicates the weight of the attention. The distance between the predictions with noise and without noise tends to show that the attention mechanism enhance the model by reducing its sensitivity to noisy data.

context through a specific training scheme. This model allows for new contexts during training. Its better structured latent space makes it easier to add new representation. This model, however, requires representation to be aggregated at inference. There is not unicity of representation of one context, which means we need to create one. Our third model resolved the challenge of unicity by adding an attention mechanism. This mechanism not only resolve some issues but also adds some stability and resilience to the model.

Attention has been widely used in the last few years (though not necessarily in the same form as we do). It has been used to fine-tuned models. Once a model is learned, only retraining where it needs to pay attention to solve a specific task is an elegant, light way to transfer knowledge. We believe this could become very useful when a new model becomes a 'universal representater' of time series, the way Convolutional Neural Network (CNN) are today a 'universal representater' of images.

CHAPITRE 4

DRIVER AND CONTEXT RECOGNITION FROM CAN DATA USING DISENTANGLED REPRESENTATIONS

Contents

Introduction

This chapter presents the industrial applications of our work. As a car manufacturer, Renault has a lot of experience in how cars function. They are also very interested as to how their products are used. Obtaining a meaningful representation (and a definition) of the driving style has long been identified as one of their long-term goals.

Driving style has been studied for several decades. Until very recently, it was either seen through the lens of road safety or as a marketing issue to target specific segments of the population. In the last few years, the emergence of Deep Learning (DL) models and the generalisation of data collection, other use cases have been envisioned. One of the issues is that driving style does not have a unique definition. Depending on the literature, there are multiple ways to define it. We therefore had to find a way to define it for the scope of this work.

We want to combine the skills of Renault with the CAN data with data driven approaches to obtain a representation of the driving style. The CAN bus is a standard of the automotive industry that is used to allow components of a machine to communicate without using a central computer. Introduced in the '80s, generalised in the '90s and now present in every car, the CAN is a wealth of information whose possible uses have only been scratched in the surface. We want to explore whether it is possible to obtain rich representations of drivers and contexts only from this data.

This task can be seen as user modelling in the context of driving. We want to obtain representations of the driving contexts that allow us to generate CAN data. These contexts are here limited to two : driving task and driving style. One of the challenges we face is that it is very hard to evaluate the generated data. The nature of CAN makes it very hard to connect it to trajectories.This is mostly due to the partial nature of the CAN data that prevents direct connection to the real world because of the lack of information about the surroundings of the car. In our case, it was made even more difficult by the quantification of the signal we collected. This is why we had to rely on auxiliary tasks of classification to ascertain its quality.

This chapter is organised as follows : in the section 4.1, we present the related work on driving behaviour. We first show the challenges that obtaining a proper definition poses. We then present the different methods that have been used to describe driving style, from questionnaires and self-reported measures to more objective, data-driven approaches sometime using Artificial Intelligence (AI). Some of the presented data driven approaches rely on CAN data, however, there are several other possibilities that were explored such as accelerometers, eye tracking, radar... We finish by presenting an overview of the many applications in which a representation of the driving style could be used.

In the section 4.2, we present the work we did with CAN data. We start by explaining in more details what is the CAN. We then present the protocol we used to collect the data and the limitations and challenges it poses. We also present the adapted version of the model presented in section 3.2 we use. Finally, we present the different experimentations we performed on this data.

4.1 Driving Behaviour Assessment

Driving style is a long-standing preoccupation. The successive trials to define and capture driving style shows that it has mostly been seen as a safety concern. Though the safety of drivers is a possible application of our work on learning to represent the driving style and task, it is not the main focus. Our goal is to obtain a representation of the contexts of driving which include the driver. We first present the historical way driving style was measured, through questionnaires and subjective measures, mostly for safety purposes. We then present the data driven approaches to driving style measurement and the previous work using artificial intelligence to obtain a representation. Finally, we present the many possible use cases where a representation of the driving style can be helpful or even necessary.

4.1.1 Challenge for a definition

The first thing to note about driving style is that there is no formal definition upon which everybody agrees. Depending on the authors, articles on the subject may use the same words to refer to different (but closely related) concepts. As an example, **4421155** defined driving style as 'the attitude and way of thinking towards the driving task' while for **6957822** it is 'the way the driving task is accomplished' which is based on how the vehicle is operated (steering wheel, pedals...). Despite this lack of commonly accepted definition, it is widely accepted that driving style is influenced by many factors, some interior to the drivers (state of mind, habits, physical abilities...) other exterior (road, time of day, weather...).

According to **review**, the main environmental factors are traffic, season, weather, road types and conditions, sight distance and light condition and other users (cars, pedestrians, cyclists...). The internal, human factors, identified are age, gender character, behaviour,

risk taking, familiarity with the vehicle, driving experience and conditions (drugs, fatigue, stress...).

Most of these indicators are difficult to measure. Because driving style was for a large part only considered as an issue of road safety, most of the tools that exist are skewed toward this goal. A recent change in the approach of driving style is the availability of data. The growth of data-driven approaches in many domains as well as the increasing availability of cheap sensors and storing device has changed the mindset. As it can be assumed in western countries that a vast majority of drivers will own a smartphone, many applications have been created. The different methods used to describe and define driving style are presented in the rest of this section.

As we could not find a widely accepted definition, we decided to come up with our own. We felt that the definitions used in behavioural studies were too oriented towards drivers' intents and state of mind for a technical-data-driven approach. The definitions used by the recent data-driven studies are very diverse and were mostly categorising drivers and not using individualised representations.

We made the hypothesis of a mapping between driver and driving style. To each driver a driving style, to each driving style a driver. We knew this hypothesis to be probably too strong to fit the reality but it seemed like an acceptable approximation. Learning to drive is akin to learning automatisms. The set of automatisms forms a set of driving habits which leads to a driving style (different from person to person). Our approach aims to learn a fine topology of drivers not starting with this set of habits and going 'bottom-up' but instead starting from the real driving behaviour.

4.1.2 Using Self-Reported Measures

Ever since road safety became a central issue, the public awareness of the harsh consequence driving can have, has increased. Many studies have shown that, once the infrastructure is secured and the car safety sufficiently enhanced with modern features, the main cause of accidents is human error [**TAUBMANBENARI2016179**]. When focusing on human factors that are involved in car accidents, studies have shown that socio-demographics (gender, age, experience...) and personality traits (impulsiveness, sensation seeking, desire for control...) have as much impact as driving skills, attitudes and behaviour [**vaiana2014driving**].

For socio-demographic impacts, the statistics clearly show that male and younger drivers are more likely to be involved in car crashes. In the USA, in 2018, there were twice as many males than females killed in car crashes [**nhtsa**]. Similarly younger drivers are overrepresented in the number of car accidents representing 23% of accidents and only 10% of the population. While other factors can explain these numbers (such as alcohol or drug consumption), this tends to show that driver behaviour is an important cause of accident.

Because this issue has become a major public concern in the last few decades, several tools were developed to quantify and define driving style and how it relates to accidento-

logy. Despite years of research and the importance of the subject, no international standard of conceptualisation or measurement exists [**review**]. The most used tool for driving style assessment remains self-reported measures. Starting in the '80s, several questionnaires were designed targeting different aspects of the driving behaviour. The two most commonly used are the Driver Behaviour Questionnaire (DBQ) [**parker1995behavioural**] and the Driver Behaviour Inventory (DBI) [**dbi**]. Both were first invented and then tested for a few years before being refined. Though they both measure driving style, they have different focus. The DBI focuses on the driver stress while the DBQ is focused on the frequency with which drivers 'engage in different types of aberrant driving behaviours'.

DBQ The DBQ was originally a 24-item questionnaire based on a taxonomy of aberrant behaviour by **parker1995behavioural**. It classes human errors into three categories, each is assigned a score to evaluate a driver.

The three categories are

LAPSES : result of inattention or absent-minded behaviour that can be annoying, but should not pose a threat to anyone safety (e.g. taking the wrong exit).

ERRORS : mistakes due to poor judgement calls or failure to see and/or analyse an important element of the context that can endanger the driver, their passengers or other road users (e.g. not checking the mirrors, not seeing a pedestrian...).

VIOLATION : deliberate break of traffic regulation or socially acceptable behaviour (e.g. speeding, excessive change of lane, bumper-to-bumper driving...).

These categories have different psychological origins, which is the reason behind this taxonomy.

The DBQ was first evaluated with its power of prediction for accident involvement, i.e. if it predicts accurately which drivers have been involved (or will be involved) in a car accident. This questionnaire has since then been refined several times and has several versions. The most classical one has 50 items but the number of items can vary from 10 to 112. The refinements are mostly to create subcategories in each category or to change the categories is some way. For example, **aberg1998dimensions** split the lapse category into errors of inattention and errors of inexperience. Other works, however, seems to show that the distinction between error and lapses is not obvious and that a better split would be 'general errors', 'dangerous errors' and 'dangerous violation' [**blockey1995aberrant**]. There are also 'local' version that adapts to 'culture-specific aberrations' [**danish**].

DEWINTER2010463 conducted a meta-study on 174 studies to investigate the relation between violation score and error score with criteria such as self-reported accidents, recorded accidents, age, gender... It has shown that violation score is highly correlated to crash rates and that error score is also correlated to accidents.

DBI Just like the DBQ, the DBI has been refined and declined in many versions [**dbi**]. It also relies on self-reported data, however, the focus is on the stress the drivers are feeling. The questions cover large areas related to driving behaviour : 'demographics, vehicle use,

accidents, traffic violations and convictions (and attitudes toward them), health, work and personal problems, mood, emotions and attitudes towards driving and other road users'. Depending on the purpose of a study, the answers can be analysed in several ways.

The goal is to evaluate the stress felt by the driver in different situations and how it affects their driving. This stress is usually measured according to six factors (some studies suggest other factors) : driving aggression, driving alertness, dislike of driving, general driver stress, irritation when overtaken, and frustration in overtaking.

Many other such questionnaires exist, like the Driving Style Questionnaire (DSQ) [**french1993decision**] or the Multidimensional Driving Style Inventory (MDSI) [**TAUBMANBENARI2004323**]. Though useful for evaluating safety and driver state of mind, they are very time consuming to evaluate a large number of drivers. Moreover, even though they take into consideration the fact that people can be mistaken (or lie) they sometime lack accuracy.

This is the reason more and more initiative are launched to use more subjective measure to assess driving style. Recently, **MARTINUSSEN201482** studied the relation between the DBQ (the 24-item 'base' version) and objective measures. This research uses a data-driven approach to evaluate the subjective measure of the DBQ. Amongst other interesting results, it showed that driver with high violation score drive faster, have more sudden acceleration, larger standard deviation of steering wheel angles, change lanes more often and spend more time in the left lane. All these indicators point to more aggressive driving style and could explain the correlation with the risk of accident. Indeed, it has been shown in different studies that speed is an important factor in both accident rate and accident severity [**nhtsa**].

This is one example of the new approaches used to evaluate driving behaviour. The rise of the approaches is driven by the apparition of cheaper, more accessible sensors.

4.1.3 Using Objective Measures : Towards More Data Analysis

In the last decade or so, several studies have researched how to use more objective measures to describe driving style. **review** made a very thorough review of them and the different language they use to describe the concepts associated with driving style. The sensors used in these studies illustrate the reason behind the emergence of these studies. Sensors have become much cheaper and in our occidental part of the world, smartphones (and the Global Positioning System (GPS) and inertial units that come with them) are everywhere.

sous-sous-section 4.1.3.1 will present the different sensors that have been used to describe driving style. sous-sous-section 4.1.3.2 will then present how they can be fused and combine to produce higher-level features. Finally, sous-sous-section 4.1.3.3 will present a review of the algorithms that have been used to describe driving style using these sensors and their higher-level features.

4.1.3.1 Sensors and Fusion

There have been a few sensors used for driving style study. Because different sensors capture different parts of the driving style, the choice of sensors is often highly influenced by the aim of the measuring [**review**].

The most used sensors in studies are smartphones. They have the advantage of being cheap, in the sense that they are widely used in everyday life. Most people are expected to own one and therefore they add no additional cost. Smartphones are used for their GPS, accelerometers and camera. The first two give insights about the position and movement of the car while the camera is used to add information about the surroundings of a car to the other information. The second most used sensors are Inertial Measurement Unit (IMU). They are now relatively cheap sensors and they contain accelerometers and gyroscopes and sometimes a magnetometer. They give measurement according to three directions : pitch, roll and yaw. The IMU does not give any information about the surroundings of a car. The same can be said for the CAN data. The CAN data is the information that transits on the CAN bus. It contains all the information that is sent from one part of the car to another. Therefore it contains information about pedals and wheel position, acceleration, speed...[1]. CAN data only contains information about the inner workings of the car, it offers no perspective on its surroundings. As such, it has been used to study driving style, though mostly for driver identification or classification.

Information about the surroundings of a car can be crucial in driving style. Indeed, if an abrupt braking is detected, knowing that it is due to the sudden braking of the preceding car instead of the style of the driver is an important information. Unfortunately, capturing the surroundings of a car is not easy. Cameras have been used but they raise issues about privacy. On the other side, RAdio Detection And Ranging (Radar) or Light Detection And Ranging (LiDAR) offer much more privacy but come at a steep price. Both Radar and LiDAR are used to detect and measure the position and speed of objects. Radar has been used for about 150 years and uses electromagnetic waves. LiDAR is much more recent (around 60 years) and uses lasers to achieve the same goal.

All of these sensors only give low-level information. They are usually not used directly into algorithms, but rather fused or processed at a low-level to obtain higher-level features.

4.1.3.2 Combination of Measures for Higher-level Features

The combination of low-level measures into higher-level features can be classified into two categories. The first one creates new measures mostly related to cameras and computer vision. The low-level inputs are used to compute new features. The second one is related to event detection. By fusing low-level measures, higher-level events can be detected. This can be done a posteriori or in real time.

The most commonly used calculated measure is the Percentage of Eye Closure (PerClos). It measures the percentage of eyelid closure over time. It allows measuring 'droops' as well as blinks. It was developed to detect drowsiness at the wheel by **perclos**. Other

1. As this the data we used in our experiment, we will go in more details to explain it in the section 4.2.

measures such as AVECLOS (percentage of time the eyes are fully closed during a one-minute interval) exists for the same purpose [**bashperclos**]. However, PerClos seems to be the most widely used. All these measure requires a camera (usually infrared) to be pointed towards the face of the driver.

Other measures can be made with this sort of camera such as Eye-Tracking Signal (ETS) [**ets**]. ETS tracks the movement of the eyes to detect alertness or head movement and position that are usually also used to detect drowsiness. There are also some studies using the same camera that are based on the detection of the frequency of yawns to detect drowsiness [**yawn**].

Some measures can also be computed or measured with the use of a camera pointing towards the front of the car. For example, lane following is often done using a forward pointed camera. Distance keeping to the car ahead is also an interesting measure that can be computed from such a camera. This detection can also be computed from LiDAR or Radar information.

Newer cars are often equipped with forward-facing-camera for the purpose of some Advance Driver Assisance System (ADAS). This camera is often used to detect signs (usually regarding speed limits to remind drivers of the current speed limit). When used in combination with accelerometers, the forward-facing camera can be used to classify braking according to the surroundings. If the camera detects an obstacle or a turn, it can 'explain' the braking. This kind of automatic annotation is at the limit between fusion and event detection.

The line between the fusion and event detection is fine because detecting braking and classifying it as sudden or not requires more than just fusion. For example, LiDAR or Radar can be fused with braking information to detect braking due to an obstacle ahead. Most of the time, however, they cannot be used to detect turns in the road.

The automatic detection of overtaking is clearly event related. It requires the detection of a change of lane and the detection of another vehicle ahead. Both these detections can be achieved either with a camera or with the combination of multiple sensors. For change of lane, accelerometers can be used to detect the sudden lateral changes. For the detection of the other vehicle, LiDAR and Radar are the usual alternatives to computer vision.

The detection of vehicles ahead of the ego-car is also used to measure the reaction delay of drivers, which is sometimes associated with events.

All of these events and measures are usually not computed as an end but as a means to an end. Though many fusions and event detection are made with AI, the results are usually passed on to other algorithms that use them to analyse driving style.

4.1.3.3 Driving Style Analysis : A Classification Issue

In the overwhelming majority of cases, the analysis of driving style is done through classification [**11** ; **14** ; **15** ; **KEDARDONGARKAR2012388**]. There can be anywhere from 2 to 7 classes. The usual 2 classes are 'aggressive' and 'calm' with a frequent addition of a 'moderate' class [**19** ; **HEV**]. A popular 4-class classification contains 'very bad',

'dad', 'good' and 'very good' [**39**]. Most of the classifications create a progression from aggressive to mild, except for one that uses 'reckless', 'anxious', 'angry' and 'patient' [**TAUBMANBENARI2004323**]. One attempt was made to use a score between -1 and 1 instead of classes [**51**].

The only other occurrence of score instead of classification to describe the driving style is related to fuel consumption [**13**; **20**]. The goal is then to attribute a score of efficiency with regard to fuel and/or power to a driving style.

Another line of work that is related to driving style is driver identification. Usually performed to detect theft, it often use CAN data to detect abrupt changes in driving behaviour [**MARTINELLI2020102504**; **wakita2006driver**]. This is a bit different from our work because the goal is to directly identify the driver and not obtain a representation of their driving behaviour.

Historically, most algorithms were based on rules [**13**]. These techniques typically define thresholds and events that are then counted to determine driving style. Their advantage is to be extremely explainable. On the other hand, they force to limit the number of parameters that can be used which impaired their accuracy. To be able to use more parameters, they have been combined with fuzzy logic [**39**]. Fuzzy logic maps can replace the extremely complex rules that are needed when using many variables.

More recently, techniques of Machine Learning (ML) have been used. The first were decision trees, probably because they remain almost as explainable as a rule-based methods [**30**]. From there, random forests became the natural evolution [**51**]. Decisions trees and random forest remain widely used to classify driving style and/or drivers [**MARTINELLI2020102504**] because they are simple to train. Other supervised models are used such as Bayesian models [**35**], Support Vector Machine (SVM) [**42**] or Markov models [**33**] (especially Markov Chain [**55**]). More recently Neural Network (NN) have been used to classify driving style **7078931**. Depending on the chosen input signal (accelerometers, CAN data, GPS, camera...) the approach can vary. The use of Bayesian models can leverage the statistical characteristic of driving while Markov Models are best suited to represent driving as a sequence of control steps. NN have shown promises using all kinds of input signals but they can require some computer heavy pre-processing because of the great variety of input signals.

One of the challenges faced by these algorithms is the lack of supervision. As driving style is not clearly defined, not all researchers used the same measures. Moreover, it is not always clear which class a driving sequence should belong. This is why, several attempts have been made to use unsupervised algorithms.

The most commonly used unsupervised methods are clustering algorithms such as k-means [**30**]. The latter is particularly used in conjunction with Dynamic Time Warping (DTW) [**14**]. Some attempts were made using Gaussian Mixture Model (GMM) to detect classes of driving style [**35**].

In all these cases (with self-reported measures or more subjective ones), the driving style is not individualised. Instead of obtaining a unique representation per driver, each

driver is categorised, it is a classification. A few categories of driving style are defined (sport, eco, aggressive...) and each driver is categorised according to some factors. What we aimed to do is different. We want to capture driving style into a representation that can then be used for other tasks. This representation should contain information about the aggressiveness of the behaviour, but also other concepts.

4.1.4 Possible Use Cases of Driving Style Representation

As shown in the previous sections, driving style has long been only seen through the lens of road safety. More recently, however, many other possible use cases of driving style were identified, some regarding fuel consumption, others having to do with autonomous driving. The Driver Assistance System (DAS), which mostly focused on safety, slowly evolved in Advance Driver Assisance System (ADAS). Though they can have safety components, ADAS can be designed for other purposes such as driver comfort or fuel consumption reduction. From the beginning of DAS, one of the goals was to make drivers aware of their potentially dangerous actions [**das**]. At first, it was mostly alarms to increase drivers' attention and alertness. Nowadays, the systems are more complex and can even take control of the car in some cases (e.g. lane-keeping assistance, forward collision warning, emergency braking...). The ADAS systems of a car can be focused as much on safety as on drivers' comfort (such as automated lighting), or even mostly focused on comfort (e.g. adaptive cruise control, traffic detection and rerouting...)

Most of these systems do not require information about the drivers' driving style. However, some of them could be enhanced with the use of driving style. We identified many applications where a representation of driving style could be useful. We classified these applications in four categories : driver mental state assessment (that is closely related to road safety), driving style enhancements, exterior services and autonomous cars.

4.1.4.1 Driver Mental State Assessment

Drowsiness (also called hypo-vigilance) has been identified as an important cause in crashes for many years [**nhtsa**]. As such, the detection of drivers' drowsiness in cars has been the focus of many research studies. To detect such drowsy behaviours, three different sorts of data have been used : some are based on biomedical data from the driver (e.g. EEG, EKG...), some use data from the car itself (e.g. steering wheel angles, acceleration, velocity...) and finally some use image analysis techniques and computer vision on images of the drivers (usually their faces). Some studies use two (or even three) of these approaches. The advantage of physiological approaches is that they are very effective as the information can be read directly. However they are also extremely intrusive, expensive to maintain and often impracticable in real-life settings (they require electrodes to be attached to the driver). On the contrary, using data coming from the car is cheap and easy to do. However, any contextual information that influences driving (such as road condition, weather, road direction...) would not be present. Finally, the use

of camera and computer vision has received a lot of attention in the last few years. The main advantage is that most indicators of drowsiness (head position, eye activity...) can be seen on the drivers' face. Though it can be seen as intrusive, the generalisation of the dashboard with driver facing cameras in modern cars has made it quite acceptable.

Intrinsically drowsy behaviour can be hard to define because it varies between individuals. The driving style could be added to these techniques so that drowsy behaviour is instead detected as the deviation from the normal behaviour.

In the same spirit, driver distraction is more easily detected as a deviation from the standard behaviour. Driver distraction is distinct from driver drowsiness as the driver is not hypo-vigilant but instead focusing on a task different from driving (such as discussion, texting...). These problems are different because the solution is much simpler for distraction than for drowsiness. For distraction, a reminder usually fixes the problem.

Finally, driving while under the influence (often of alcohol, sometimes other mind-altering substances) is the most prominent cause of lethal accident **[nhtsa]**. Once again, the driving style could be used to detect and alert in cases where the driving behaviour is too different from what it usually is ; or if it is detected as erratic and/or dangerous.

4.1.4.2 Driving Style Enhancement

The previous use cases are used to enhance the safety of road users both inside and outside the vehicle. There are other applications that aimed at enhancing the behaviour of the driver. We have identified two main applications of drivers' behaviour enhancement : eco-driving and driving Electrical Vehicule (EV) or Hybrid Electrical Vehicule (HEV). There are a few other applications of drivers' style assessment that we present afterwards.

Eco-Driving Amongst the different sorts of ADAS not mainly focused on road safety that have appeared in the last few years, the 'eco-driving' kind is one of the most popular ones. This ADAS has been declined in many different applications. Depending on the application, 'Eco' is not necessarily well defined but usually refers to a driving style that would reduce the gas consumption thus leading to less pollution and cheaper transport. Many such ADAS can run on simple smartphones, using its accelerometers. They usually give drivers grades on different metrics (such as rapidity of acceleration or braking) which can inform the driver on how to progress. Cars manufacturers have also started to incorporate such applications in cars with the advantage of having access to many internal measures inside the cars (such as immediate consumption and number of rotations per minute for the motor). Combined with GPS information, this allows two things : better context for the car (the altitude gives a lot of information for how much acceleration there should be at some specific point) and comparison. Some applications allow drivers to compare their gas use with other drivers that drove on the same road, leading to a gamification that can be beneficial (as long as the competition is not to do the worst possible).

In this use case, a representation of driving style can be used to direct a driver into a 'better' driving style. For example, if you have a few representations of virtuous styles, you can push a driver towards one of these styles (usually, the closest one) to improve their performance.

Driving EV or HEV An application to improve driving style is particularly useful in EV and HEV, where the optimal driving style (in terms of energy use) is quite different to the one most drivers are used to [**HEV**]. In this case, an ADAS could teach a driver new to this type of vehicle, how to better drive them. Indeed, there are some habits that greatly improve the autonomy of a battery that are not necessarily known to all EV/HEV drivers. For example, it is better for the battery life expectancy to avoid fully charging or fully discharging it. Though most systems would prevent users from doing so, it is probably better to recharge the battery before it drops below 30% capacity and not charge it over 80%. Here, car manufacturers have to balance the autonomy of one charge of the battery (how much distance the car can drive before recharge is needed) and the life expectancy of the battery (how long before it needs to be changed). One possible use of driving patterns would be to predict how much power will be needed until the next day to anticipate the charge. By predicting how much power the next trips will require, the battery can be charger soon enough but not too soon.

More related to driving style, when driving EV and HEV, it is important to anticipate acceleration and braking to optimise energy use and battery recharging. In an HEV that is not Plug-in Hybrid Electrical Vehicule (PHEV), the braking is the only way to recharge the battery and must therefore be especially looked after (during descent, when arriving at a stop or red light...). Another approach is rooted in the idea that driving style is unlikely to change from just a few tips given by the car. Some work has been done on how to best optimise the torque demand between the two motors of a HEV according to a classification of the driving style. In this case, the idea is to adapt the car to the driver and not the opposite.

General Driver Assessment More generally, some applications can assess the performance of drivers. In this case, a representation of driving style should be able to predict many of the indicators. By measuring the performance of drivers on different metrics, not necessarily focused on eco-driving, some applications help them improve. For example, some focus on the comfort of the passengers (for children of fragile people) and point out to the driver how they could improve it. Some applications focus on helping inexperienced (often young) drivers to develop good habits both for security and driving style. Finally, some other applications focus on the wear and tear of the car and how to preserve some components of the car (such as tyres or brake disks).

4.1.4.3 Exterior Services

All of the previous applications are focused on the driver. There are other use cases in which the driving style can be used outside of the car. There are numerous such use cases, too many to be exhaustive. Here are the most important ones that we have identified.

Public Health Extending on the work done on detection of hypo-vigilance, there are some studies that aim at using the car as a diagnosing tool for sleeping problems. If a driver is too often found to be drowsy at times where there is no reason, it can be a sign of a disease or disability (such as sleep apnoea). In this case, although it can be used to improve the safety of road users, it is also an ADAS that can be used for public health. This application is currently researched but it faces big obstacles, the main one not directly related to driving and cars : data privacy.

Predictive Maintenance In this use case, the goal is to get individualised predictions of when to change specific parts of the cars. Depending on how the car was used, some parts can be damaged sooner than others. This allows manufacturers to sell a service where car owners get a tailored, individualised support and maintenance. This is especially useful for car manufacturers in the case of leasing (long-term rentals with or without the option to buy). Indeed, in this case, the maintenance costs fall back to the car manufacturers which gives them incentive to optimise it. It also brings much information to manufacturers about the durability of their parts. Recently, for example, with the democratisation of EV and HEV, there are some questions about the durability of the batteries used. This kind of application allows manufacturers to collect information on the long-term (as this technology is relatively recent, there is not much hindsight on the subject).

In this case, a representation of the driving style could be used to predict how much a driver will wear and tear some specific part of a car. It can also be useful in the case of leasing to determine a premium.

Simulation of Part Resilience Predictive maintenance is useful after a car is built. However, during design, some adjustment can be done. An important part of car design is that each part has to work for many years. During design, decisions have to be made regarding the durability of parts (which has to be balanced with cost). Simulations as well as real use are used to determine the resistance of each part. If the resistance of a part can be adapted to driving style, cars could be adapted to each user (or more likely category of users). If one wears the brake disks faster than average but the tyre slower than average, adjustment could be made.

Driver Identification In this case, the car itself could be used to detect who is driving. This could be applied in insurance where a driver style would be evaluated before deciding on a rate. From a representation of drivers, the risk associated with them could be computed and their rate adjusted. This could also allow insurance to be sure of who is

driving a car. This can be especially useful in case there is an accident or for 'pay as you go' type of insurance.

Identification of drivers through their driving style has also been used in fleet management. Using simple accelerometers and other cheap sensors (instead of mounting expensive cameras and other surveillance equipment) to evaluate the states of mind of their drivers can significantly reduce the fleet management costs [**fleet**]. This also allows them to score drivers in order to help them improve. It can also serve as a warning when a driver is in a state (tired, angry...) where they should not drive. As mentioned above, this can be used to perform predictive maintenance, which for a company owning a fleet of vehicles, can significantly reduce costs.

4.1.4.4 Autonomous Cars

Finally, an identified use case for a representation of driving style is in the autonomous cars. These types of cars (that do not require drivers as they drive themselves) are still experimental. However, in the last few years, several experiments on very large scales have been launched, with self-driving cars used on specific portions of roads and specific contexts. One of the larger (and longer) one is the one launched by Waymo (formerly Google Car) in the San Francisco area. In the last 5 years, the company has had an entire fleet of autonomous cars, driving in the streets of San Francisco, Mountain View and the roads in the surrounding area. The first reported accident of one of their cars was on the first of March 2016. Since then many more (of different severity) have happened. Though none were fatal, they have been widely publicised and discussed in public.

What is interesting to note is that all these accidents were investigated and, in each case, the self-driving car was not to blame. Each time, it was human error that caused the accident. However, in many cases, though the autonomous vehicle strictly followed the regulations, its behaviour was not predictable for a human (for example, stop at an intersection where there is no other vehicle or pedestrian in sight, or slow down at a specific place where humans usually don't). These accidents (most of them minor) showed the challenge to road safety a mixed of self-driving and human-driven cars could pose. This will be especially dangerous as the number of autonomous vehicles increases. One way to mitigate the issue is to parametrise the behaviour of the self-driving cars. Without allowing it to break the rules of traffic, giving a self-driving car a more 'human' driving style could make it more predictable for humans and therefore reduce the risk of accident. Moreover, it could be interesting for the people inside the autonomous vehicle (most likely its owner and/or renter) to be able to change the behaviour of the car. They could make it drive in a sportier way or on the contrary more prudently depending on the context (return from the hospital, sleeping infant...).

We showed that driving style, though it has been extensively studied is still not very well defined. Depending on the aim of the study, some definitions can change. At first only studied as a road safety problem, it has evolved and could be used in many possible use cases. Our work differs from the other presented above by the fact that we want to

obtain a universal representation. Instead of limiting to one aspect of the driving style, we want our presentation to encompass them all.

4.2 Our Data Driven Approach Based on CAN Data

Our goal remains to learn disentangled representations of the contexts. We choose our contexts to be the driving style and the driving task. We know that many more aspects can affect driving. We decided to focus on the driving style. The idea is that the driving task should include all other aspects. The driving task was fairly easy to define and we chose to use GPS location and the weather (despite knowing that other aspects could be considered). Each specific part of the road is a different driving task. The driving style was not as easy to define, as we showed in the previous section. That is why, we made the hypothesis of a mapping between drivers and driving style. Each driver is mapped onto a driving style.

To construct these representations of contexts, we want to use Renault's strength : its deep knowledge of the inner workings of cars. This knowledge allows us to understand the data interior to the car. This data, the CAN data, will allow us to describe driving style without using questionnaire or subjective measures but rather a data-driven approach.

In sous-section 4.2.1, we describe the data we used in our experimentations. We first present a description of CAN data, followed by a description of the experimental setup that allowed us to collect the data. Finally, we explain the challenges these data poses to use.

In sous-section 4.2.2, we present the experimentations and models we used on this data and their results. We first performed several preliminary experimentations to explore our data and discover what information is contained in it. We used classification to find out if information about the style and task are present. We then adapted the model presented in the chapitre 3 to learn representations of the style and task through generation and forecasting.

4.2.1 Our Data

We want to part with questionnaires and subjective measures and use data driven approaches to describe the driving style. Therefore we need data that contains the relevant information. Because Renault has a world expertise on the subject, and unique insights for years of designing it, we decided to use CAN data. We will first present the CAN data. We will then present the experimental setup we used to collect it. We will finish by a presentation of the challenges posed by the collected data.

4.2.1.1 What is the CAN ?

The CAN is a bus that is used in automotive engineering to allow controllers, sensors and other devices to communicate with each other without using a centralised computer. In cars, for example, it is the means to transfer messages from the wheel to the wheels. Almost all the information that allows a driver to drive the car transits on this bus. When pressing a pedal for example, the information about how much it is pressed transits on the CAN and goes to the engine control unit that computes the change to be made to the engine. This information is sent in a trame format that first needs to be decoded.

The frequency at which each information is sent on the CAN depends on the information. For example, some measurements are sent at $100\,\mathrm{Hz}$ while others are made at $1\,\mathrm{Hz}$. A device, usually called 'dongle', can be connected to the car On-Board Diagnostics (OBD) port to read the trames of the CAN.

Some technical choices presided over the choice of the dongle. Some devices allow for real-time collection through the use of SIM cards, while others store the data on the device for later reading. Depending on the device's ability, the trame can be read at different frequencies. There is an incredible amount of information that transits at all times on this bus and our goal is to use this information to take a data-driven approach to describe driving style.

4.2.1.2 Experimental Setup

The first choice we needed to make was the 'dongle'. Because we did not need real-time measurement, we chose a device that would store the data on board. Though this choice had unforeseen, unfortunate results, it was, at the time, the best technological decision we could make. The second choice we had to make was the selection of specific channels from the CAN. Because our ultimate goal is to learn disentangled representation of tasks and style, we want to capture information pertaining to both factors. The issue is that the data has to be stored on the device and measurement made every $10\,\mathrm{ms}$ rapidly take a lot of space. We therefore had to select channels to be captured. One hundred and thirty measurements from the CAN were selected as well as the 14 measurements of the GPS.

The experimental setup was done with multiple purposes for cost reason. The main drive of the experiment was to measure how driving style impacts the wear and tear of the car. We were able to add some constraints to the protocol in order to obtain both task and driving style supervisions. The collect took place on tracks in Aubevoye used by Renault to test and experiment cars. The tracks are meant to mimic different contexts such as mountain roads, highway roads, city roads... An approximate map of the tracks can be seen in Figure 4.1.

Using six brand new cars (2 Clio, 2 Traffic and 2 Mégane), 117 professional drivers drove for 90000 km (each car for 15000 km) on the tracks over a period of two months. The drivers drove the cars in shifts of 8 hours, with an 8-hour pause for each car every 16 hours of driving. For half the shifts (divided along cars), the drivers were asked to use

FIGURE 4.1 – A Map of Renault Experimental Tracks

an 'Eco' driving style that is designed to optimise the gas use and for the other half a 'dynamic' driving style that is more focused on speed and sporty behaviour.

For each shift, we therefore have the location of the car, as well as the identity of the driver and information about the weather and the exterior temperature. Because one of the goals of this collect was to research the impact of driving style on the wear and tear of cars, measurements of some components of the car throughout the collect wear also made (brake disks, thickness, oil quality...).

4.2.1.3 Challenges

The data we collected from the CAN during this experimentation presents many challenges. The first one is that the data is extremely noisy (as many real-world data). This is partly due to faults in the CAN itself where the information can sometimes be lost or corrupted. The many redundancies inside the car are there to correct such problems but it does not necessarily transit on the CAN.

Moreover, the collecting dongle is an embedded system, and as such, it has some flaws. For example, to limits the storage space used by data on the dongle, some of the measurement was quantified and compressed. A measurement that should vary every $10\,\mathrm{ms}$ ($100\,\mathrm{Hz}$) is actually compressed and stored with the same values for may 'ticks' because the dongle estimate that it did not change sufficiently since the last tick. This was an unforeseen drawback of the dongle we chose. An illustration of this problem is presented in Figure 4.2 where we compare the recording of two different dongles of the transversal acceleration reported on the CAN (resampled at $10\,\mathrm{Hz}$). It is obvious that one of the dongles (the one we chose for this work) does not measure as accurately as the other.

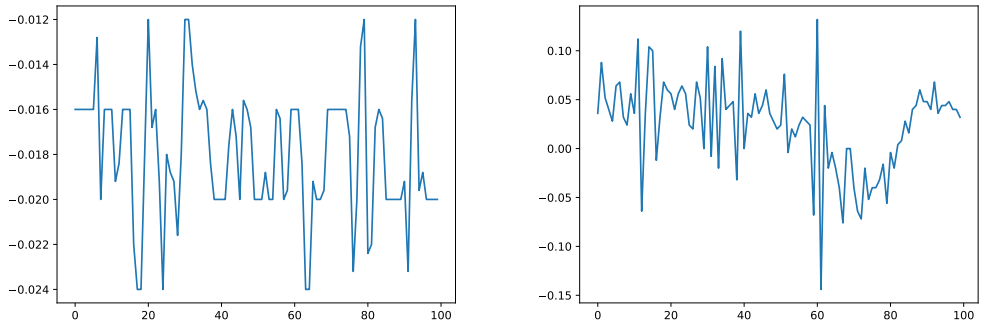

FIGURE 4.2 – A comparison of the measurements of two different dongles of the transversal acceleration reported by the CAN. They correspond to different cars and different trajectories. The figure on the left corresponds to measurements by our dongle. When compared with another dongle (on the right), the quantification problem we faced becomes obvious.

Depending on the quantities studied, the recording frequency can be very different. The GPS protocol dictates a maximum of 1 point per second (1 Hz) while the Revolutions Per Minute (RPM) is always recorded at 100 Hz. This also leads to some challenges of synchronisation.

One of the main challenges CAN data poses is to connect the raw data to the real world. Having access to data from the CAN is not sufficient to know the real trajectory of the car. There are many exterior factors such as weather, state of the road, wear and tear of the tyre that can affect the trajectory of a car. From the same movement of brakes and pedals, depending on these factors, the trajectory can differ. Moreover, the physical laws that preside over the movement of cars are extremely complicated and not all information necessary to compute them are recorded on the CAN. Indeed, we believed that integrating the accelerations and speeds recorded in the CAN would allow us to recover the trajectories. However, it turned out to be impossible in practice. The results of the integration were very different from the GPS trajectories and did not match the shape of the road. We performed a few other experimentation, using yaw speed (that also transits on the CAN) but the results were not much better.

We then tried to learn a model that would connect the CAN to a trajectory. We faced a problem linked to supervision. The GPS is at best accurate to within one metre (and average around 10 m in precision) which, on a road four metres wide, is not very accurate. The GPS position can also be obtained at most every second. The supervision is therefore not very good. Moreover, the quantification of the data that resulted in loss of high-frequency variations probably impacted the model. Though we believe it is possible to learn such a model, it was deemed outside of the scope of this work.

For this work, it means that the generation of CAN data is difficult to evaluate. We have no way to know if the generated data is realistic.

4.2.2 Experimentations

For this work, we used data from all the drivers from one type of vehicle (the Clios). We used GPS information to limit the data on a specific part of the tracks, the one meant to mimic the highway. We focused on shifts that happened in days where the weather was dry (it was not raining or snowing and the road was not wet and/or icy). We first performed preliminary experimentations using supervised classification to explore the data and discover whether or not the required information about contexts was contained in the CAN. We then used a model similar to the one presented in section 3.2 to learn disentangled representations of two contexts : the driving style and the driving task.

4.2.2.1 Classification

We performed a few preliminary experiments. The goal was to verify that CAN data did contain information about the driving style and task. We created a first dataset to discover driving style through classification. It only has one supervision : the drivers' identity. We then created a second dataset that has two supervisions : one on the drivers' identity, the other on the tasks. This dataset was also used in the generative experiment presented in the next section.

Driver Identification The first task we tried to solve was the classification of windows of driving data into the ID of the driver. This was our preliminary work to determine whether the information about the contexts (in this instance the driving style) was contained in the CAN data we had. We pre-processed the data to create windows of 10 seconds of driving. Each of the CAN channel was resampled to 10 Hz which resulted in 100 time steps windows. We selected 30 drivers that drove using 'Eco' style and 30 drivers that drove using 'Sport' style. For each driver, we collected 1000 windows. The created data was split into train/valid/test (70%/15%/15%) and normalised using mean and variance.

A visualisation of the dataset can be found in :fig :6o$_c lass_b inary@cref$

:fig :6o$_c lass_b inary@cref$

:fig :6o$_c lass_b inary@cref$

:fig :6o$_c lass@cref$

:fig :6o$_c lass_b inary@cref$

:fig :6o$_c lass_b inary@cref$

:fig :6o$_c lass@cref$

:fig :6o$_c lass@cref$

:fig :6o$_c lass_b inary@cref$

4,2,2,2,2 :fig :6o$_c lass@cref$

4,2,2,2,2 :fig :60$_c lass_b inary@cref$

:fig :60$_c lass@cref$

:fig :60$_c lass_b inary@cref$

:fig :60$_c lass@cref$

:fig :60$_c lass_b inary@cref$

:fig :60$_c lass_b inary@cref$

?? . For two channels of the CAN (longitudinal and lateral acceleration), violin plots show the distribution of the normalised values. Figure 4.3 shows the difference of distribution for 'Eco' and 'Sport' style. Though the values are in the same range, the distributions do look different. Figure 4.4 shows each individual driver. The numbers 0 through 29 are the 'Eco' drivers while the other are the 'Sports' drivers. The differences are more subtle than in the previous case. This comfort us in the hypothesis that driving style can indeed be found in CAN data.

We performed two classification tasks on this dataset. The first one attempted to classify these windows only into 'Eco' or 'Sport' style. The second one classified the windows into 60 classes (one per driver). We tested three classifications models to perform this task. One used Long Short-Term Memory (LSTM), one Multi-Layer Perceptron (MLP) and the last one XGBOOST. The LSTM used as input the sequence of length 100, the two others use histograms of 10 bins (one histogram for each CAN channel). We found through a feature selection study that the channels that gave the best results were longitudinal and lateral acceleration. The baseline used in these experiments is a linear SVM.

	LSTM	MLP	XGBOOST	Linear SVM
Binary style	**0.82**	0.71	0.73	0.54
60-classes style	**0.21**	0.08	0.01	0.02

TABLE 4.1 – Our results (in Mean-Squared Error (MSE)) for driving styles classification. The task with only two styles seems to be the easier and the results are as expected : The LSTM model achieves the best results while the MLP and XGBOOST achieve similar, quite good results. The baseline performs slightly better than random.

This preliminary model was encouraging. Our LSTM model achieves great results for classification. Therefore we moved to finding more than one context.

Driver and Task Identification The second supervision we created was on the driving tasks. Using GPS information, we designated 15 spots using the tracks map that were of interest (specific turns or interesting parts of the road). Instead of taking random windows of 10 seconds of driving, we created our windows based on the distance to each of the designated spots. This allowed us to create 11870 windows of 10 seconds of driving (using the same resampling as previously, we had 100 time-step time series) distributed (not uniformly) along 79 drivers and 15 tasks. Of the 79 drivers, 42 were driving using 'Eco'

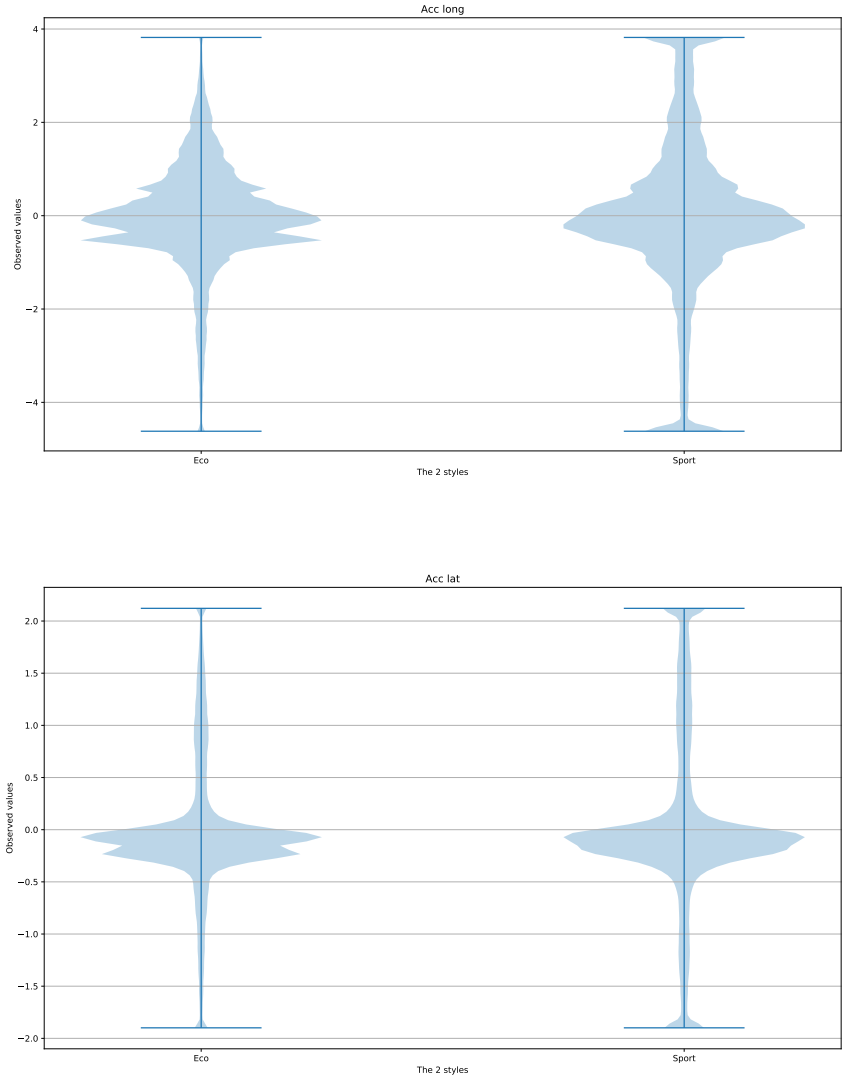

FIGURE 4.3 – A visualisation of our dataset of driving windows with supervision on style. This shows the distributions of normalised values (with 2% outliers removed) for two styles 'Eco' and 'Sport'.

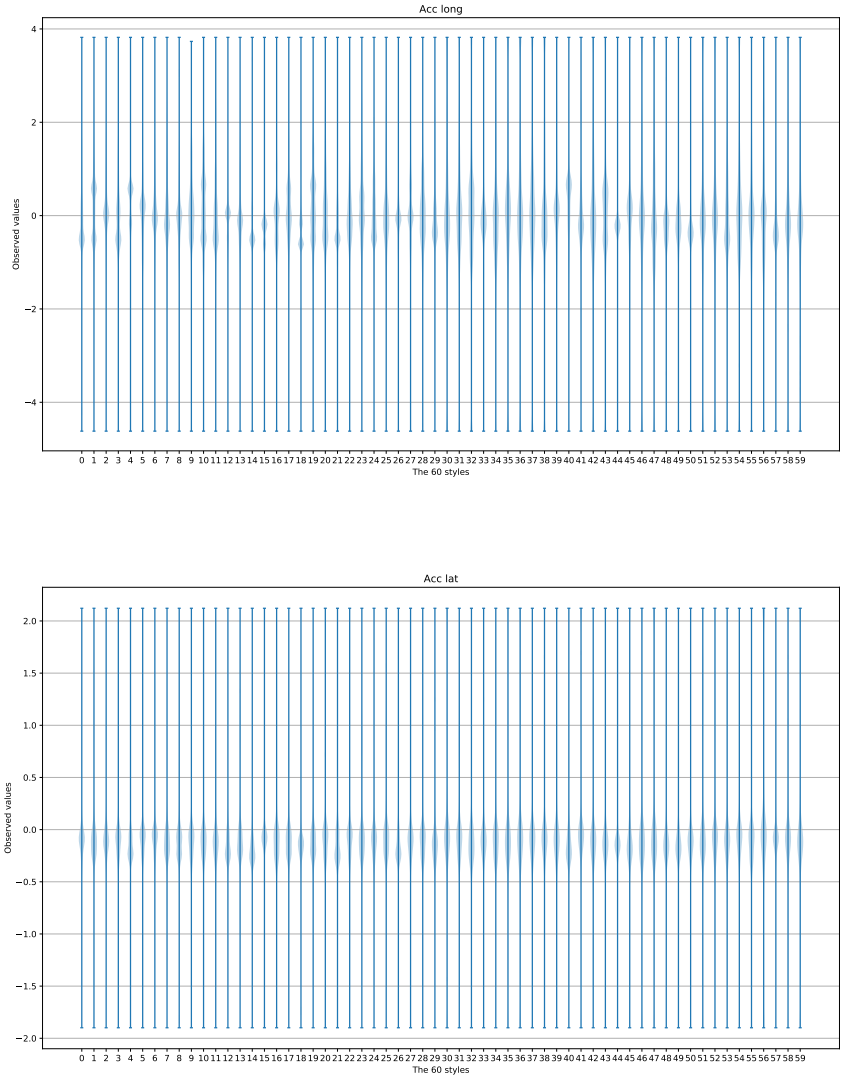

FIGURE 4.4 – A visualisation of our dataset of driving windows with supervision on style. This shows the distributions of normalised values (with 2% outliers removed) for the 60 styles. The plots numbered 0 through 29 are the 'Eco' drivers while the others are the 'Sports' drivers.

style and 37the 'Sport' style. This dataset was split into train/valid/test (70%/15%/15%) and normalised using mean and variance.

A visualisation of this dataset is not as easy as the previous one. The :fig :15tasks@cref

:fig :15tasks@cref

:fig :15tasks@cref

:fig :79styles@cref

:fig :15tasks@cref

:fig :15tasks@cref

:fig :79styles@cref

:fig :79styles@cref

:fig :15tasks@cref

4,2,2,2,2 :fig :79styles@cref

4,2,2,2,2 :fig :15tasks@cref

:fig :79styles@cref

:fig :15tasks@cref

:fig :79styles@cref

:fig :15tasks@cref

:fig :15tasks@cref

?? shows visualisation of some aspects of this dataset. Figure 4.5 shows the distribution of normalised values for each task for only two classes of style : 'Eco' and 'Sport'. The even-numbered plots show the distribution for 'Eco' drivers while the odds numbered ones show the distribution for 'Sport' drivers. It shows that the tasks clearly have different distribution, with different ranges. The difference between the two styles are not as visible but still quite clear.

The Figure 4.6 shows the distribution of values for each driver on our first task (numbered '0' and '1' in the previous figure). The plots numbered 0 through 37 are the 'Eco' drivers, the others the 'Sport' drivers. The reason there are not 79 violin plots is that not all drivers in our dataset drove on this task. Unlike in the previous chapter, where our dataset could be seen as a matrix of time series, here we have a 3D tensor. Two of the dimensions are the contexts. The third is the number of time a specific driver drove on the specific task. This adds complexity for our model as these trajectories are not identical. From the same two contexts, there can be several possible reconstructions.

We now have two classification tasks : one to classify the driving task, the other the driving style. Just like previously, we attempted two tasks on the driving style, one binary classification (sport/Eco) and one by driver (79 classes). The XGBoost model can only perform one classification at a time, so we trained one for the task and one for the style. The MLP and LSTM can perform both classifications at the same time. We compared using two different models for each classification task (trained separately or not) or using the same model with only the last layer to classify for each task and found that there

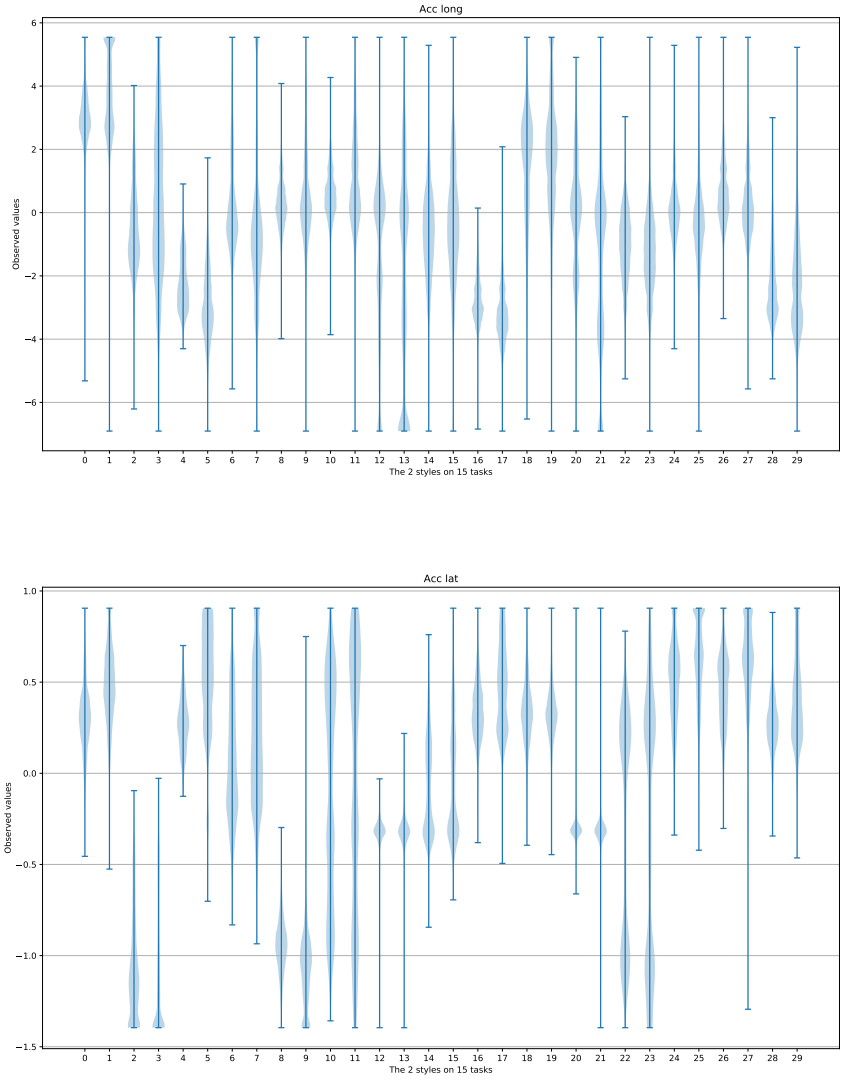

FIGURE 4.5 – A visualisation of our dataset of driving windows with two supervisions : tasks and style. This shows the distributions of normalised values (with 2% outliers removed) for each of the 15 tasks. The 'Eco' and 'Sport' styles are shown side by side for comparison. The even-numbered plots correspond to the 'Eco' style, the odd-numbered one to the 'Sport' style

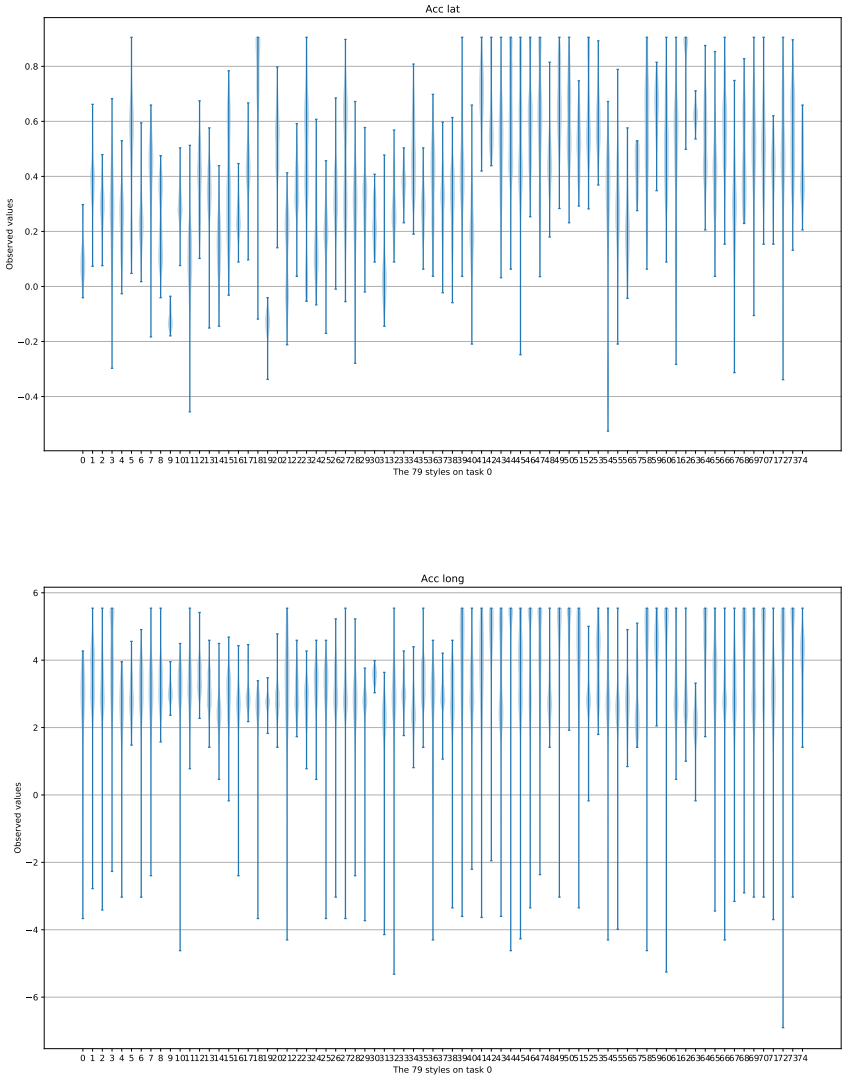

FIGURE 4.6 – A visualisation of our dataset of driving windows with two supervisions : tasks and style. This shows the distributions of normalised values (with 2% outliers removed) for one of the tasks (the one corresponding to '0' and '1' on the Figure 4.5). Plots numbered 0 through 37 are the 'Eco' styles, while the other corresponds to the 'Sport' style.

was not much difference in the performance. The second option (classifiers sharing all layers except the last one) is less computationally intensive. It is also closer to our goal of learning representation of driving style and driving task. This is why we chose this option. Once again, the baseline in these experiments is a linear SVM.

		LSTM	MLP	XGBOOST	Linear SVM
Binary	Style	**0.80**	0.74	0.77	0.61
	Task	**0.93**	0.57	0.86	0.56
60-classes	Style	0.17	0.07	**0.31**	0.03
	Task	**0.91**	0.71	0.86	0.56

TABLE 4.2 – Our results (in MSE) for style and task classification. The task classification seems to be much easier with even our baseline doing much better than random. The 60-style classification task is much harder. This time the XGBOOST model clearly outperforms the other models.

These results are very encouraging as they show that a large part of the information about driving style and driving task are indeed present in the CAN data. Though the classification using neural network could give us a representation of the contexts through the last layer before classification, we want to use a model specifically to obtain disentangled representations of each context.

4.2.2.2 Disentangled Representations

Our last task was performed using the same dataset as previously. This time, the goal is to obtain disentangled representations of the driving task and style. The model we decided to use is a simpler version of the one described in the chapitre 3. Inspired by an Autoencoder (AE) model, it consists of two encoders (one for each context) and a decoder. To avoid auto-encoding, instead of reconstructing the input directly, the output is decoded from the mixing of the representation of two examples. For example, from the trajectory of driver 1 on task 1 and trajectory of driver 2 on task 2, reconstruct the trajectory of the driver 1 on task 2. As explained in section 3.2 this forces the model to learn disentangle representations.

This model was designed for learning representations with a generative or forecasting supervision. In this use case, generation is not an easy task to supervise. Because the quantities we work with are not easily linked to real world trajectories, it makes it very hard to assess the quality of the generative process. There are several ways to take a left turn and a driver will not take the exact same trajectory each time they pass. It is hard to quantify how those little changes in trajectories affect the accelerations we work with. As such, and though we use it in training, it is not clear what the significance of the MSE

(done channel by channel) is. The first measurement we thought of is to compare the distribution of values for a specific combination of tasks and drivers. However, comparing distributions is not an easy task and is often used more as a qualitative assessment. The quantitative measurements we designed are based on classification. In addition to training two encoders and one decoder for the model, we added two classifiers which, from the latent representations, must learn the classes (of tasks or driver) the original input examples belong to. This model is illustrated in Figure 4.7.

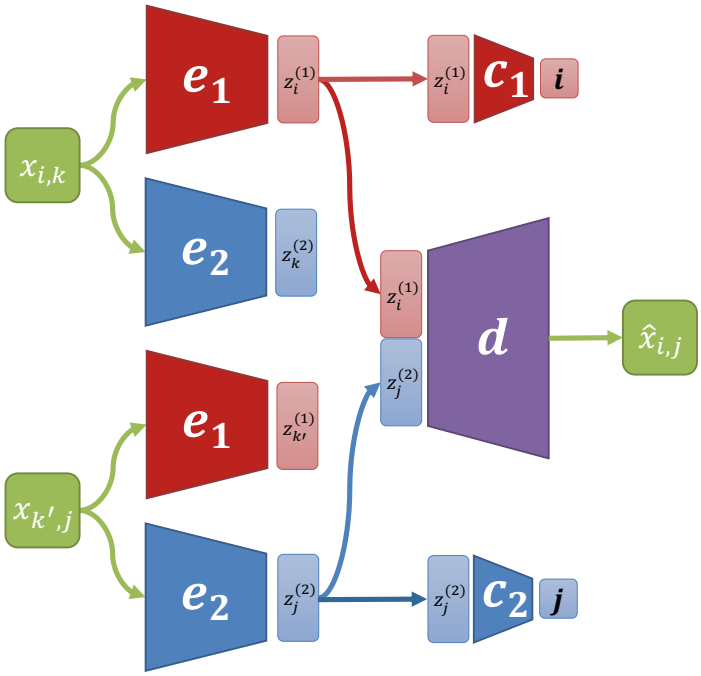

FIGURE 4.7 – A representation of our model. It is an adaptation of the one presented in section 3.2.

This model was learned end-to-end, with both encoders, both classifiers and the decoder trained at the same time.

The classification results (presented in Tableau 4.3) are very close to the one presented above. This tends to show that the learned representations of contexts contain all the information about driving style and task contained in the original time series.

The issue of evaluating the generated time series was not resolved. We have obtained a loss of reconstruction that allows comparison between models but it is not clear what this loss means. The other solution we used was to visually compare curves and distributions to measure the generative power of our model.

4.2.2.3 Discussion and Future Work

The experiment of generation and forecasting was less conclusive than with the previous datasets. This is mostly due to the challenge posed by the evaluation of the generative process. There are several possible explanations for this fact. The first one is that,

		LSTM	Classification LSTM	Classification XGBOOST
Binary	Style	0.78	0.80	0.77
	Task	0.92	0.93	0.56
60-classes	Style	0.16	0.17	0.31
	Task	0.91	0.91	0.86

TABLE 4.3 – Our results (in MSE) for the classification of our learned representations. The results have barely changed from the previous experiments, which are shown here for comparison. This tends to show that our representations do contain all the information the original time series did

as explained above, our data is very quantified. It is possible that the loss of the finer, high-frequency, information greatly impacts our results. There are too many cases where using the previous value as prediction is a very strong baseline. It is also possible that the two contexts we chose (task and style) are not rich enough. The time of day, that greatly affects the lighting conditions, was never taken into account. Moreover, the weather supervision we used is not very specific even though it can also affect the lightning condition. We believe that with more contexts and better supervision, we could achieve better results.

As it stands, we have gained experience in using CAN data in ML models. We have obtained representations of contexts of driving (specifically tasks and drivers) only from CAN data (and supervision from GPS). We believe that these representations could be used quite directly in some tasks. The latent space of representation of tasks could be used to create profiles of tasks. The nice clusterisation it presents could allow for models to be trained to extract real world information from them.

The overall result of these experimentations is that information about the driving style and the driving task are indeed contained in the CAN. The first thing we noted is that task is clearly easier to find than style. Differentiating between binary driving styles is also quite easy to perform. Driver identification, however, seems harder. We believe this could be due to the fact that our drivers are professional and therefore drives with very close, efficient styles. This hypothesis should be tested in the real world. We believe that 'normal' drivers should be more easily identified.

Renault launched another collect of data, this time aimed at employees (not professional drivers). With their consent, the CAN data and GPS of their car is collected in real time on a platform. We believe this data could be used to prove our hypothesis about driving style. Preliminary studies have shown that the challenges faced in this experiment are different. The environment is not at all controlled. The drivers don't drive the same cars nor do they drive on the same roads. When focusing on style, the car model is extremely important. As the physics is different from one model to another, the measurements also

are. The first models trained on this data learned not to recognise the driver but the model of the car. Some more experiments need to be conducted to erase the car model from the representation of the driving style, for example by using representation censoring. Another problem lies in the lack of supervision on the task. More precisely, the lack of overlap with tasks for the drivers makes the transfer of our model directly to this data difficult. We do not currently have perspectives on how to overcome this difficulty except adding another explicit context.

4.3 Conclusion

In this chapter, we presented the studies on driving style and task we conducted. The first step was to read and summarises the literature on the subject of driving style. We realised that this concept is not well defined. It has been measured in different ways in the last decades. At first, it was mainly defined by a few categories drivers would be classified into, according to self-reported measures and behavioural questionnaires. It is only recently that the use of more objective measures of driving has generalised. The democratisation of smartphones and the decreasing cost of many sensors has led to several studies and applications using data driven approaches to driving style. Building on these studies, we took a new approach by using data internal to the car. By using CAN data, we combine the skills of Renault in building and designing cars with the power of DL approaches. The first step towards this goal was to collect data in a controlled environment. By using brand new cars driven by professional drivers on very well-known tracks, we aimed at eliminating the as many factors of variations as possible. The collected data was then used in a few experiments. This very raw data presented us with a few challenges, some inherent to the data, some linked to technical choices. Despite these obstacles, we showed that the use of CAN to represent driving style and task data is a promising avenue for research, though other complementary source of data should be considered to gain insight on contexts. We managed to learn representations of styles and tasks that turned out to be quite good.

CHAPITRE 5

CONCLUSION

5.1 Where we began

This work set out to perform context modelling from data, with the hypothesis that users may be a context not very different from other contexts. Using Deep Learning (DL) approaches, we aimed at learning disentangled representations of contexts. These representations could then be used in many different tasks such as data augmentation or completion. One of the motivations for this work is firmly anchored in an industrial research problem for Renault : how to model driver behaviour from data generated by the car. Renault has gained a lot of experience in simulation using very complex physical models. However, data-driven approaches are relatively new to the industry. This work was exploratory, to find out what could be achieved using Controller Area Network (CAN) data. The historical approaches of such rich but extremely noisy raw data were statistical and we wished to improve on such approaches by using DL models.

5.2 Where we are now

For this work, we decided to use a disentanglement approach. In this framework, the learned representations have to be connected to different real world, meaningful concepts. In our case, each context was to be disentangled from the others. We achieved two main contributions :

Our first contribution consists of a model that can learn disentangled representations of supervised contexts from time series. We have shown that as long as such labelled contexts exist in the dataset, our model can learn their representations sufficiently well. During inference, it can generate missing data from unseen combination of contexts. It can also generalise to generate from contexts only seen after training. This model was incrementally enhanced to improve its performance and widen its use cases.

Our second contribution was closer to our industrial research problem. We analysed and used CAN data to create a proof of concept. We showed that it contains enough information about driving behaviour and driving task so that Machine Learning (ML) models can learn to disentangle such contexts. Though this data is, in essence, noisy and unaware of anything but the car, we showed that it could be used to disentangle driving task and driving style. Our first models learned to classify short windows of driving into tasks and drivers. We then improved them to also learn representations of both contexts.

We did not know much about the subject when we began our work thus we had to make some hypothesis. Although these hypotheses turned out to be mostly correct, our results show that refining them is likely to improve the results.

5.3 Perspectives

This work has opened many perspectives, in both research and applied directions. We present first the two extensions of our work we began exploring : a weekly supervised model and an Reinforcement Learning (RL) approach. We then discuss the other possible perspectives this work has opened. We finish by a presentation of the perspectives of our industrial problematic with an interesting use case : the nudge.

Weekly-Supervised

In the short term, our work has some very direct applications. The work presented in chapitre 3 could easily be transformed by a transportation company into a monitoring board that would detect anomalies in the system very early (and predict the effect on the entire day). The same company could use this work to run simulations for the opening of new stations and predict the number of people that would use it. We could also improve these predictions by using other information (such as geographic and demographic data) to better understand new factor value.

Another, quite direct, extension of our work would be to use the model on datasets that have more than two contexts. As datasets with this much supervision are hard to come by, an unsupervised version of the model could be devised. With one (or more) well-labelled contexts, the model could extract other information about unknown (or unlabelled) contexts.

We started to investigate in this direction. Using negative sampling and triplet loss, we developed a model that can learn good representations of both contexts by using only supervision on the other context. The first experiments we performed on the STIF dataset presented in sous-sous-section 3.3.1.1 are very promising. We are, however, faced with the same problem of having to measure the quality of representations. We started to explore a few research directions in this area. We are investigating using measures such as class of the nearest neighbours and other clustering algorithm directly in the latent space to assess the quality of our learned representations. This extension is still under investigation but advanced enough that we have a few results.

Reinforcement Learning (RL)

Another natural extension of our work is through Reinforcement Learning (RL). Because of the sequential nature of our data, RL seems to be a good research direction for our problem. Indeed our data can be seen as a sequence of driving actions (that can all be

extracted from the CAN) and states (that are at least partially observed from the CAN). Not only is RL the main learning framework used to solve sequential problems, but it is also known to produce good representations. RL models an agent that interacts with an environment and receives a reward. At each time step, an action is taken that leads to a new state that may be only partially observed. In our case, if we consider the task of driving to a designated spot, a state could be the position of the car, its current heading and its velocity while the actions would be the CAN values needed to drive the car to the correct destination. RL requires the agent to learn a representation of the states. The representation must encompass all the aspect of the states, so that the correct action is taken. This is why RL approaches are known to produce good representations. The decision-making process requires to be aware of all important aspects of the environment.

Instead of using classical supervision, RL uses a reward function. The goal is to optimise the reward. To simplify, when the agent arrives in a desirable state, the reward is positive, when it arrives in a bad state, the reward is negative. In our case, hitting an obstacle would have a big negative reward while arriving at the destination would have a positive reward. To prevent the car from just not moving at all, a very small negative reward would have to be given at each time step. The objective is to learn an optimal policy which maximises the cumulated expected reward (a balance between immediate reward and long-term strategy). In our case, as the positive reward is only given when reaching the arrival, we need to work with cumulative rewards. This allows the policy to balance the non-positive rewards received when going to the destination with the positive reward obtained at the end.

Unfortunately, this approach based on the optimisation of a single objective usually results in policies that are somewhat unrealistic. This is mainly due to the problem of reward shaping. Defining a reward function in cases like the game of Go or Atari is relatively easy (win/loss). However, for more difficult or complex tasks such as autonomous driving, the process is not as easy. Because the reward is so difficult to describe, the resulting policies are often disappointing. With a reward as simple as described above, the chances of obtaining a car that even remotely respect the rules of the road are almost zero.

As our goal is to learn representations of context, it does not necessarily require to learn good driving policy. Our approach of contexts modelling through RL is instead to learn policies that are parametrised by representation of the contexts. Just like before, we focus on the representation of two contexts : the task and the style.

We decided that the best way to learn disentangled representation of our contexts was to have a two-part model. We first learn a policy A that is parametrised by the task and must learn to accomplish the task. This policy takes as input the observation of the environment and the representation of the task factor and outputs a decision. The second part, called B, is parametrised by the style. It takes as input the observation, the decision of the policy A and a representation of the style and outputs a modified decision that corresponds to the chosen style. The latent spaces of representation of the style and the context are learned together with the policies.

This model is illustrated in Figure 5.1

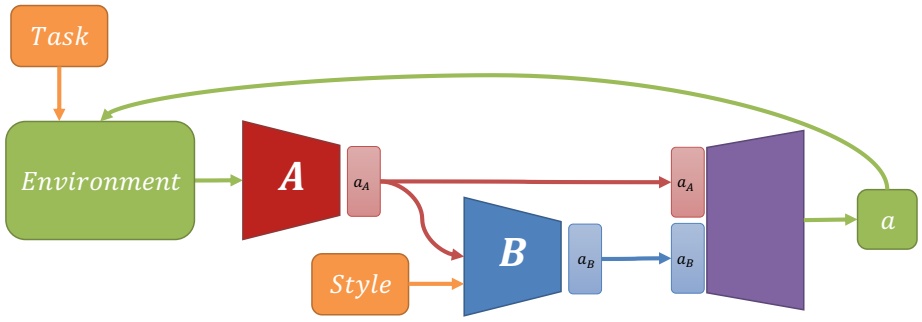

FIGURE 5.1 – Our proposed model where A learns a general policy and the task context and B is a modifier that learns the style context

The main advantage of this model is that A and B can be learned through different methods. For example, A could be learned from a classical RL setting by interacting with an environment while B is learned with data from real use with Imitation Learning (IL). IL is a subclass of RL where the agent learns by copying the behaviour of 'experts' that have been recorded solving the tasks.

This decoupling of A and B could be leveraged in several ways depending on the available data and environment.

We started to investigate 3 possibilities of datasets/environments that could have well-defined contexts to experience with our two-part model. We called the contexts style and task to keep the terminology of our chapitre 4.

Lunar Lander The first idea we had was to modify an already existing RL environment to fit it to our goal. Lunar Lander is a popular gym environment **[gym]** where the agent must learn to land a (simplified) lunar lander on a platform. This task is quite distant from our driving task but this environment is well researched and can be used to experiment with some ease. It is also a simpler task that can be used to explore out models and refine them.

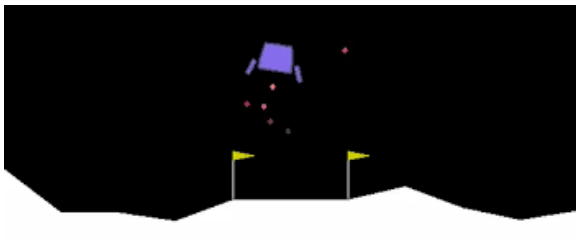

FIGURE 5.2 – Print screen of an Agent Playing Lunar Lander

The agent only controls the engines of the lander (left, right and main engine). The reward is given according to the position and velocity of the lander. There are two versions of the environment, one discrete, the other continuous.

In the classical gym environment, the lander has always the same form and must always land in the same location. As the environment is open-source, we decided to slightly modify it to obtain different contexts.

For example, the form of the lunar lander can be altered to almost any polygon, the location of the landing platform can be changed as well as the size and position of the legs. All of these changes allow for similar but different tasks to solve and therefore a context for the task.

To create different context of styles, we modified slightly the reward function. The original reward function is a weighted sum of different characteristics of the lander (position, speed, heading, gas consumption...). By giving more weight to the speed or the use of gas, we can generate different contexts.

Square This is a gym environment we developed to better match our goal of having two different contexts. Closer to our actual task than Lunar Lander, it is also much simpler than a driving simulator. In this environment, an agent is placed inside a square of 5 by 5 : the field. The field contains obstacles that are 1 by 1 square that can only be placed on one of the 25 squares that make up the field. The agent is a disk of diameter 0.2, centred on its coordinates. The agent has a velocity and a heading that makes it move around on the field. The agent starts in the middle of the bottom left square of the field. At each time step, the agent takes an action that can modify its heading or its velocity. The agent then moves from a distance equal to its velocity in the direction where it is heading. If a movement puts the agent outside of the limits of the field or overlapping with an obstacle, the movement is modified so that the agent bounces off on its sides without losing velocity. An episode terminates when the agent reaches the upper right (pink) square of the field. The action space of this environment can be either discrete or continuous.

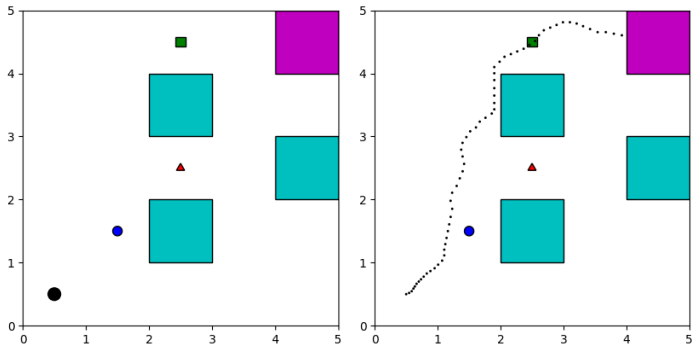

FIGURE 5.3 – A field from our environment with three obstacles (left) and a trajectory on this field (right)

We create different context of tasks by changing the location of the obstacles. The start and end do not change, only the cyan squares that represent the obstacles change.

To add another context, we also added three marks : one red triangle, one blue circle and one green square. These marks are placed in the middle of one of the empty squares

that make up the field (not at the start or end nor in one of the obstacles). The agent can overlap with these marks. The 'style' is measured by the distance to the mark an agent keeps (or does not keep). An agent can 'like' green and 'dislike' blue meaning it will pass on (or close by) the green square and not go near the blue circle. This 'liking' is modelled as an exponential decrease from the distance to the mark.

AirSim AirSim is an open source car simulator based on Unreal Engine and developed by Microsoft. It is meant to offer a platform to test autonomous driving algorithms. It allows the recording of the action of the driver as well as other contextual information. We developed our own environment : car tracks that goes through different contexts, inspired by the real-world test tracks of Renault.

FIGURE 5.4 – Image of the track of our Unreal Environment (Left) and screen of a person driving on our track (right)

We developed a protocol to gather data from different people driving on our tracks. They all drove 5 turns of the track in one direction and 3 turns in the other. The first two turns are not used in the data and considered as the learning time for the driver (time to get used to the simulator and the track).

Similarly to what we did in chapitre 4, we defined each task as a segment of the tracks. It can be a turn, a straight line, a U-turn... We 'cut' the trajectories of the drivers and selected small parts of them that represent our tasks. As for the style, we made the same hypothesis that each person has their own driving style, therefore, each driver represents

a context of style.

We have not found conclusive results using RL. Despite that, we believe this is a promising research direction for learning disentangled representation of contexts.

Other possible research directions

A natural, but quite important, modification of our model would be to allow for multiscales contexts. In sales prediction for example, a big hurdle in the multi-scaled nature of context. There are scales at the product level, at the brand level, at the shop level, at the country level... Finding a way to represent and aggregate the different levels of contexts could greatly improve the quality of the predictions.

Another issue also found in sales forecasting is that the timescales are not synchronised. Though products may have the same curve in general in their lifetime, the scales (both in time and amplitude) can differ a lot. The time series representing sales of these different products are not uncorrelated. It could be interesting to detect an event in one of these series and to find a model that can spread this effect to the other related time series. Note that this would also be useful for the datasets we used in chapitre 3.

The context of the pandemic we are facing at the time of writing brings another possible (very long term) use of our model adapted for related time series. If we could learn representations of diseases and countries (as our contexts) and how the latter are connected, our model could be modified to predict the expansion of such an epidemic. This could also be used on a smaller scale for regions within a country or subregions within a region.

Industrial Applications

For Renault, a natural extension of this work would be to confirm our results on another set of CAN data. This could confirm some of the hypotheses we made as to the limitations underlined by our experiments.

On a longer term, we hope that this work could be the first step toward a comprehensive representation of drivers. Or, to be more specific, of people in the context of driving. A regular, meaningful latent space of drivers could have many applications.

An interesting use case : the nudges

One such application of user representation in the scope of our industrial research is the use of *nudges* associated with drivers' representation to better driving behaviours.

It is not clear where and when the term nudge comes from, but it was popularised by **nudges**. Richard H. Thaler was awarded the Nobel prize in economics in part thanks to

his formalisation of the nudge theory and how to apply it in economy. His work led to the development of behavioural economics. Nudges are defined by **nudges** as :

> 'A nudge, as we will use the term, is any aspect of the choice architecture that alters people's behaviour in a predictable way without forbidding any options or significantly changing their economic incentives. To count as a mere nudge, the intervention must be easy and cheap to avoid. Nudges are not mandates. Putting fruit at eye-level counts as a nudge. Banning junk food does not.'

Though it was mostly research in economics, nudge theory has been applied to many domains. The most famous example of nudge in the real world is the use of houseflies in man urinals. Though only one study was conducted, it showed that by simply etching a housefly, the Schiphol Airport in Amsterdam was able to reduce by 80% the cleaning bill of their men's restrooms. This very small change incentives men to 'better direct the flow'.

Closer to our domain, the use of nudges to improve road safety has grown during the last decade. One such example is the use of smiley faces on the side of the road that are either happy if the speed is below the limit or unhappy if the car is speeding. The presence of coloured smileys seems to be more effective than just display the current speed. More complicated examples exist such as the use of road painting to create the optical illusion that the car is going faster than it really is on specific dangerous portion of the road. Not directly linked to road safety, the Indian authorities experienced with a red unhappy smiley that would blink and buzz each time a driver would use the horn. The smiley had to be turned off manually by the driver, which was apparently so annoying that honking dropped by 61% during the experimentation.

Combined with a meaningful, regular latent space of representation of drivers, nudges could prove very useful to improve drivers' behaviours. Using known nudges from inside the car, it should be possible to push drivers toward a better driving style. This would, of course, requires the knowledge of what the results of each nudge on the driving style representation in the latent space are. This would not be easy and would probably require nudges' effects to be learned at the same time as the latent space. This could also be possible with a high enough number of drivers, so that the nudges would allow to 'jump' from one driver to the next. Using this technique, drivers could be nudged toward a better, more virtuous driving style. Virtuous or better could depend on the drivers' definition and mean more safety, less gas consumption or an increase in the lifetime of the car.

Abstract

The recent, sometimes very publicised, successes have drawn a lot of attention to Deep Learning (DL). Many questions are asked about the limitations of these techniques. The great strength of DL is its ability to learn representations of complex objects.

Renault, as a car manufacturer, has a vested interest in discovering how their cars are used. Learning representations of drivers is one of their long-term goals. Renault's strength partly lies in their knowledge of cars and the data they use and produce. This data is almost entirely contained in the Controller Area Network (CAN). However, the CAN data only contains the inner workings of a car and not its surroundings. As many factors exterior to the driver and the car (such as weather, other road users, road condition...) can affect driving, we must find a way to disentangle them.

Seeing the user (or driver) as just another context allowed us to use context modelling approaches. By transferring disentanglement approaches used in computer vision, we were able to develop models that learn disentangled representations of contexts. We tested these models with a few public datasets of time series with clearly labelled contexts. Using only forecasting as supervision during training, our models are able to generate data only from the learned representations of contexts. They even learn to represent new contexts, only seen after training.

We then transferred the developed models on CAN data and were able to confirm that information about driving contexts (including driver's identity) is indeed contained in the CAN.

Résumé

Les récents succès, parfois très médiatisés, de l'apprentissage profond ont attiré beaucoup d'attention sur le domaine. Sa force réside dans sa capacité à apprendre des représentations d'objets complexes.

Pour Renault, obtenir une représentation de conducteurs est un objectif à long terme, identifié depuis longtemps. Cela lui permettrait de mieux comprendre comment ses produits sont utilisés. Renault possède une grande connaissance de la voiture et des données qu'elle utilise et produit. Ces données sont presque entièrement contenues dans le CAN. Cependant, le CAN ne contient que le fonctionnement interne de la voiture (rien sur son environnement). De nombreux autres facteurs (tels que la météo, les autres usagers, l'état de la route...) peuvent affecter la conduite, il nous faut donc les démêler.

Nous avons considéré l'utilisateur (ici le conducteur) comme un contexte comme les autres. En transférant des méthodes de désenchevêtrement utilisées en image, nous avons pu créer des modèles qui apprennent des représentations démêlées des contextes. Supervisés uniquement avec de la prédiction pendant l'entrainement, nos modèles sont capables de générer des données à partir des représentations de contextes apprises. Ils peuvent même représenter de nouveaux contextes, qui ne sont vus qu'après l'entrainement (durant l'inférence).

Le transfert de ces modèles sur les données CAN a permis de confirmer que les informations sur les contextes de conduite (y compris l'identité des conducteurs) sont bien contenues dans le CAN.